(2)

AUTOMATIC DOCUMENTATION METHODOLOGI FOR SOFTWARE MAINTENANCE

TECHNICAL REPORT

TECHNICAL SOLUTIONS, *Inc.*

P.O. BOX 1148
MESILLA PARK, NM 88047
(505) 524-2154

2

AUTOMATIC DOCUMENTATION METHODOLOGIES FOR SOFTWARE MAINTENANCE

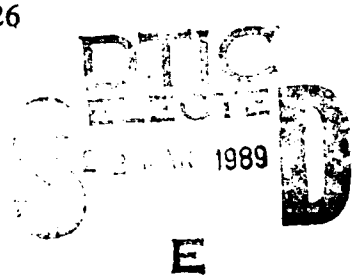
TECHNICAL REPORT

L. D. LANDIS
P. M. HYLAND
A. L. GILBERT
A. J. FINE

15 JANUARY 1989

U.S. ARMY RESEARCH OFFICE

CONTRACT #DAAG029-85-C-0026



Technical Solutions, Inc.
P.O. Box 1148
Mesilla Park, N.M. 88047

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) ARO 22935.3-EL-S
6a. NAME OF PERFORMING ORGANIZATION Technical Solutions, Inc.	6b. OFFICE SYMBOL (If applicable)
6c. ADDRESS (City, State, and ZIP Code) P.O. Box 1148 Mesilla Park, NM 88047	7a. NAME OF MONITORING ORGANIZATION U. S. Army Research Office
	7b. ADDRESS (City, State, and ZIP Code) P. O. Box 12211 Research Triangle Park, NC 27709-2211
8a. NAME OF FUNDING/SPONSORING ORGANIZATION U. S. Army Research Office	8b. OFFICE SYMBOL (If applicable)
8c. ADDRESS (City, State, and ZIP Code) P. O. Box 12211 Research Triangle Park, NC 27709-2211	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DAAG29-85-C-0026
	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

12. PERSONAL AUTHOR(S)
L. D. Landis, P.M. Hyland, A.L. Gilbert, A.J. Fine

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM Sept 85 TO Nov 88	15 January 1989	168

16. SUPPLEMENTARY NOTATION The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Given the non-trivial costs of software systems, it has become imperative to maintain software for a much longer period of time that was considered desirable in the past. Providing tools that will facilitate software maintenance helps to extend the useful lifetime of a software system.

This report discusses a research project that was directed at providing a general-purpose, automatic documentation generator that could provide both the detailed and higher-level information that maintenance programmers perceive useful.

The project proceeded in three phases. The first phase consisted of an examination of the documentation methodologies currently available, and language requirements necessary to achieve these documentation methodologies. The second phase consisted of development of the design for a documentation language which would be used as an intermediate represenation in

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

the documentation process. Finally, a research prototype was developed, in order to test the design and validate the research conclusions.

ABSTRACT

Given the non-trivial costs of software systems, it has become imperative to maintain software for a much longer period of time than was considered desirable in the past. Providing tools that will facilitate software maintenance helps to extend the useful lifetime of a software system.

This report discusses a research project that was directed at providing a general-purpose, automatic documentation generator that could provide both the detailed and higher-level information that maintenance programmers perceive useful.

The project proceeded in three phases. The first phase consisted of an examination of the documentaion methodologies currently available, and language requirements necessary to achieve these documentation methodologies. The second phase consisted of the development of the design for a documentation language which would be used as an intermediate representation in the documentation process. Finally, a research prototype was developed, in order to test the design and validate the research conclusions.

Table of Contents

List of Figures and Tables

CHAPTER 1 - INTRODUCTION

1.1 OVERVIEW OF THE PROBLEM AND FOCUS OF THIS REPORT

The ever-increasing costs of software systems motivate a desire to protect the initial investment in such a system by maintaining it for as long as possible. However, software has a limited lifetime of usefulness, because as it ages support becomes more difficult. Major factors in determining when to replace, rather than to maintain, a software system are 1) the costs associated with employing and supporting a programmer working on the code, 2) the time required to train programmers to maintain code following in-house standards, and 3) the costs associated with maintaining multiple versions of a software system over an indeterminate period of time.

Empirical results indicate that maintenance programmers consume an average of one-fourth to one-third of allocated modification time tracing and understanding the logic in the software to be modified. [Fjeldst] A significant gain in maintenance programmer productivity (hence, a reduction in maintenance costs, and an extension of the software's lifetime) should be achievable by providing those programmers with a tool that will facilitate understanding program logic.

This report presents research performed by Technical Solutions, Incorporated (TSI) addressing the feasibility and development of just such a tool. This report does not address tools for software managers or software developers; the focus is on a tool for maintenance programmers, providing them with information that will improve performance. The research discussed in this report was supported by the U.S. Army Research Office, under contract number DAAG029-85-C- 0026.

1.2 BACKGROUND AND RELATED WORK

While there are two types of documentation associated with software systems, only one type is of interest to maintenance programmers and to this research. User documentation, or information about the purpose and use of a software system, does not (and should not) address the system's internal structure and operation. It is this internal information that is of interest to the maintenance programmer.

Internal, or programmer documentation, can take a number of forms, but it has only one goal: to provide information to the programmer regarding the internal structure and purpose of the code itself, addressing such issues as the data structures and control structures employed, the static structure of the software (sub)system, any "tricks" used or shortcuts taken by previous programmers, and any other information an implementor deems appropriate. This information serves to provide an environment in which the maintenance programmer can work.

While much research is being done in the areas of various kinds of software maintenance, maintenance environments, configuration management, etc., little research is being done regarding tools for maintenance programmers. This was made evident by examining the

Proceedings of the Conference on Software Maintenance (CSM) for the last three conferences (1985, 1987, and 1988). An examination of the abstracts for all of the articles in all three of the Proceedings yields the following breakdown of topics, as shown in Table 1.2.1, below, where tools in general are the only item of interest.

TOPICS OF CSM ABSTRACTS	# OF ARTICLES	# OF PANEL DISCUSSIONS
CSM-85:		
tools	6	1
other research	22	7
CSM-87:		
tools	2	0
other research	19	6
CSM-88:		
tools	8	0
other research	52	5
TOTAL		
tools	16	1
other research	93	18

CSM Breakdown Of Topics
Table 1.2.1

As can be seen from Table 1.2.2, documentation was the topic most commonly addressed. This emergence of research regarding documentation tools signals a new awareness of the importance of software maintenance tasks, and of the role of maintenance programmers.

TOOLS	# OF ARTICLES
Software Managers	1
Non-Prototype	1
Documentation of Source Code	8
Discovery of Code Structure	4
None of the Above	2
TOTAL	16

CSM Breakdown Of Tools
Table 1.2.2

1.3 PURPOSE, GOALS, AND APPROACH

The purpose of the research reported herein was to study the feasibility of a tool to facilitate maintenance programming. Consequently, the end goal of this research was to develop a prototype tool, embodying the concepts discovered to be of interest to maintenance programmers. Since good internal documentation facilitates the understanding of code, it was determined that a documentation generator depicting static code structure, dataflow, overview, and detailed information would be the end product. With the development of this documentation generator as the final goal, we worked backward to identify other goals.

The documentation generator was to be able to accept a general class of structures from general-purpose programming languages, thus it was decided to build upon the well-understood concepts of compiler theory, structuring the tool much like a production compiler. The source code to be documented would be consumed and used to build symbol and code tables which would completely specify the source code. The symbol and code tables would then be consumed and used to generate the documentation itself.

From this process, it was determined that information was needed regarding the documentation methods available, and regarding the feasibility of various potential source languages for such a tool.

Thus, the goals for the research were:

1. Research documentation techniques specifically appropriate to the software maintenance environment, as contrasted with methods commonly used in development;

2. Summarize the programming language features that were appropriate for the class of potential source languages for the documentation tool;

3. Research the design of a general documentation language, focusing on the ability to handle a selected class of languages, using the proposed approach (structured in the three categories of preprocessor, compiler, and postprocessor); and

4. Determine if the proposed approach could be implemented to automatically transform programming language source code into documentation.

These four goals were used to identify the milestones in a multi-phase research program.

The research involved the development of automatic documentation generation in three steps: preprocess the source language into an intermediate representation (or documentation language), compile the documentation language into a parse tree (with symbol tables), and finally, generate the selected documentation from the parse tree.

The next section provides an overview of the project. Each phase: research, design, and implementation, is presented in greater detail in Chapters 2, 3, and 4, respectively.

1.4 OVERVIEW OF THE PROJECT

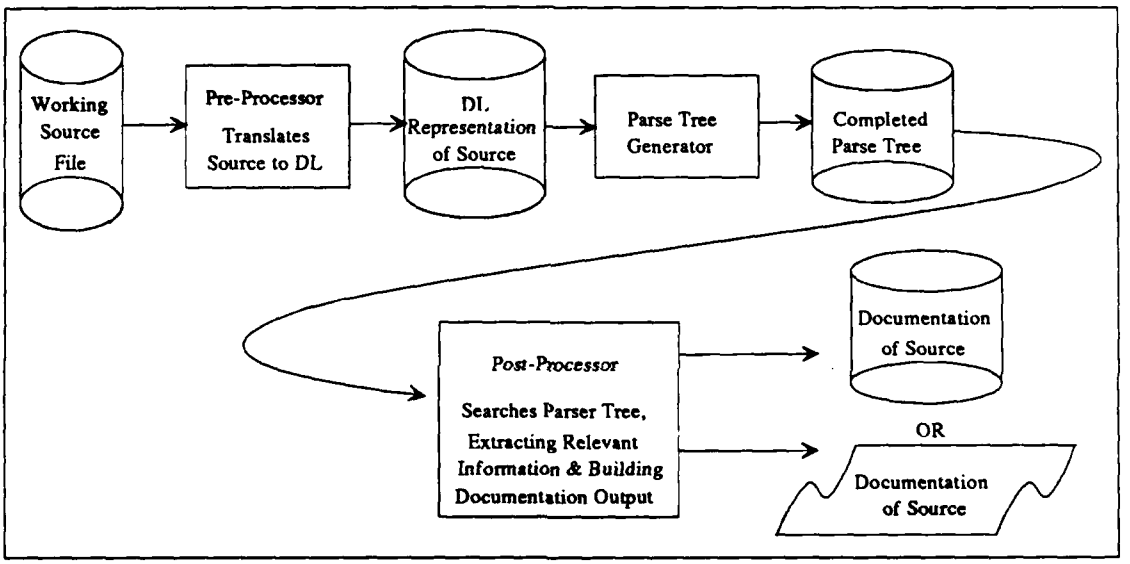
The four goals presented above were incorporated into the work plan as follows:

Phase 1: The Research Phase involved the determination of input and output requirements for the documentation generator (thus, the general documentation language). (Achievement of goals one and two, above.);

Phase 2: The Design Phase involved the design of the general documentation language. (Achievement of goal three, above.); and

Phase 3: The Implementation Phase involved implementing the design, proving that the research approach was viable. (Achievement of goal four, above.)

Figure 1.4.1 depicts the dataflow diagram for the system, or tool, that would be the result of this project. The tool was named the Parser/Documenter (P/D).



Parser/Documenter Dataflow Diagram
Figure 1.4.1

1.4.1 RESEARCH: APPROPRIATE DOCUMENTATION TECHNIQUES FOR SOFTWARE MAINTENANCE

The research began by evaluating the methods of documentation that are standard practice in industry, with a specific emphasis on their appropriateness in a maintenance programming environment. There were several evaluation criteria, including whether a given methodology could represent both detailed and overview information, and whether each methodology could represent static code structure and dataflow information.

The evaluation led to an important requirement for the maintenance tool. In order to maximize the perceived usefulness of a maintenance tool, the documentation provided must be familiar to a large class of programmers currently working in industry. Expecting programmers to learn a new documentation methodology, in addition to their other responsibilities, was concluded to be counterproductive.

Central to the research was the identification of target languages which would serve as source languages for the tool, and from which the documentation language would be developed. The selected target languages included FORTRAN, Pascal, and C, with some consideration to the other standard third generation languages: COBOL, DIBOL, and BASIC. Additionally, many of the requirements of Ada were considered, but as a whole, Ada was deemed too complex to attempt on a first implementation.

Each of the target languages was examined as to its peculiar requirements, and those requirements were compiled into a list. From the individual language lists, a summarized list of requirements was developed, reducing the conflicting requirements to special cases, where needed. This resulted in a derived/hybrid language requirement.

Consequently, nine of the most popular documentation methodologies were investigated, and the commonly used third generation languages were examined. From the set of nine documentation methods, three were selected to test the proposed approach. The primary technique chosen was Nassi-Shneidermann (NS) Diagrams, while Cross-References and Action Diagrams were chosen as secondary techniques. The class of potential source languages was reduced to FORTRAN and C.

1.4.2 DESIGN: THE NEXT STEP

The third goal of the research was to design a general documentation language that would facilitate the determination of the validity of the proposed approach. Once the requirements for source languages (input) and documentation methods (output) were identified, the design of the general Documentation Language (DL) proceeded. Data and control structures were designed to represent the attributes of the emerging DL. From the data structures, the source representation of the DL was developed. As work proceeded, it appeared that the X3J11 version of C would serve as a good base. Minimal extensions were made to the grammar, and the result was a general purpose documentation language that allowed for the retention of all the requirements.

The design was then executed "by hand": the sample DL was transformed into data structures which represented the parse tree and symbol tables to be generated by the compiler. From the data structure diagrams, a conversion to NS Diagrams was performed.

With the completion of this process, i.e., the reduction of the requirements, the manual construction of data structures, and the manual interpretation of those data structures, an implementation of the approach was deemed possible.

1.4.3 IMPLEMENTATION: THE RESEARCH PROTOTYPE

The implementation of the approach began with the DL compiler. One important consideration was to prevent any need for special documentation signaling in the original code. This required the DL to be a complete language. Documentation signaling has frequently been a requirement for automatic documentation generators. Pseudo-comments may be used as a signaling device to a documentation generator, yet these signals are buried in comments, which are ignored by the compiler.

For example, an automatic documentation "extractor" could easily be constructed to process the routine shown in Figure 1.4.3.1 to produce the documentation shown in Figure 1.4.3.2. In this example, the signaling technique used was based on the rule that certain key words are processed in a special manner, specifically, a beginning comment string, i.e., "/***" on a line by itself, up to the closing comment, i.e., "*/" on a line by itself, delimits a signal to the compiler.

```
/***
 * $Locker:  $
 * $Header: pd.1,v 2.3 88/10/21 14:05:37 ldl Exp $
 */

/***
 * File:
 *   usrlib/icsmalloc.c
 *
 * Description:
 *   To allocate ICS memory
 */

#include "icslib.h"

/***
 * Synopsis:
 *
 *   #include <ics.h>
 *   #include <icscall.h>
 *
 *   long
 *   icsmalloc( fd, size )
 *     int   fd;       // file descriptor of the ics device
 *     long  size;     // size of memory to allocate
 *
 * Description:
 *   Allocate a block of memory.  If a large enough chunk of
 *   memory is not available, but the last free block
 *   contiguous to video 0 memory is large enough if memory
 *   overflows into video 0, then that block is returned.
 */

long
icsmalloc( fd, size )
  int   fd;
  long  size;
{
  long ret;
  icscall( fd, "malloc", &ret, 1, size );
  return ret;
} /* icsmalloc */

/* EOF usrlib/icsmalloc.c */
```

An Example Routine Processed By An Automatic Documentation "Extractor"
Figure 1.4.3.1

```
|---
|
| icsmalloc                    Programmer's Reference Rev: 1.1
|
|
| NAME
|     icsmalloc - To allocate ICS memory
|
| SYNOPSIS
|
|     #include <ics.h>
|     #include <icscall.h>
|
|     long
|     icsmalloc( fd, size /
|       int  fd;       // file descriptor of the ics device
|       long  size;    // size of memory to allocate
|
| DESCRIPTION
|     Allocate a block of memory. If a large enough chunk of
|     memory is not available, but the last free block
|     contiguous to video 0 memory is large enough if memory
|     overflows into video 0, then that block is returned.
|
| SEE ALSO
|     usrlib/icsmalloc.c
|
| CAVEATS
|     None
|
|---
```

The Resulting Documentation Produced From The Routine Shown In Figure 1.4.3.1
Figure 1.4.3.2

This concludes the introductory discussion of the project. This discussion is expanded in some detail in the following Chapters.

1.5 FORMAT OF THE REPORT

Chapter 2 presents the research performed for this effort. Chapter 3 provides the design of the research prototype which would later be used to support the research conclusions, and Chapter 4 provides the implementation of the research prototype. Chapter 5 presents the conclusions of this effort. Appendix A contains the DL grammar, Appendices B and C present two examples of P/D operation, and Appendix D contains the references and a list of sources for related reading.

CHAPTER 2 - THE RESEARCH PHASE

The research phase was performed in three consecutive stages. First, the appropriate documentation techniques were determined as they are embodied by the documentation methodologies in use today. Second, the target group of languages was evaluated to determine a common set of requirements which would then direct the definition of a general documentation language for that group of languages. Third, the technical validity of the proposed approach was reevaluated.

2.1 DETERMINATION OF APPROPRIATE DOCUMENTATION TECHNIQUES

As previously stated, the end goal of the research was to provide useful documentation to maintenance programmers. For this evaluation, the mindset was that of "What sort of documentation would help me, a maintenance programmer, to achieve a better understanding of this code?". As such, this review was subjective, and thus the opinions of maintenance programmers were solicited and taken into consideration during this phase.

2.1.1 EVALUATION CONSIDERATIONS

The languages under consideration in this research were all of a general purpose nature. Thus, there was very little that could be done to "precast" the form of the documentation, i.e., force the results to take a particular, universally-acceptable form. For example, if the language being documented was instead a special purpose mathematical evaluation language (such as Maxima, Mimic or Midas), it would be relatively easy to cast the form of documentation provided.

General purpose languages, on the other hand, are used to specify anything from simple expressions to complex simulation models, to real-time process control, or to the compilers for the languages themselves. The varying purposes motivated the need for a variety of documentation outputs to be addressed.

In order to achieve a workable set of documentation methods, a set of evaluation considerations or acceptance criteria had to be developed. It was determined that the following criteria were of importance to the problem:

- Whether each methodology was capable of representing static code structure or dataflow information, and whether this information was available in detail, or as an overview;
- Whether that same static code structure or dataflow information required any specialized output devices;

- Whether each methodology would work well (if at all) on code not designed and developed with it;
- Whether each methodology could be learned easily; and
- Whether it would be feasible to extend the functionality of the methodology.

Thus, the first step in evaluating the documentation methodologies was to review the standard documentation techniques currently used in industry. Once selected, these techniques were each subjected to the acceptance criteria so as to constrain the initial set of postprocessors to those that would be the most useful to a maintenance programmer. Finally, each selected documentation form was examined as to whether extensions would make the method more useful, and the final form of the documentation was formalized.

2.1.2 DOCUMENTATION TYPES EVALUATED

In order to demonstrate the documentation methods evaluated, Figure 2.1.2.1 is an "original" code fragment which was then represented in the various methodologies. Of the nine following subsections, which discuss a documentation methodology, eight contain the methodology's representation of its code segment. These subsections include presentations of Pretty Printers, Warnier-Orr Diagrams, Jackson Diagrams, Flowcharts, Nassi-Shneidermann (NS) Diagrams, Pseudocode, Action Diagrams, Higher Order Software (HOS) Charts, and Cross-Reference/Usage Listings.

```
1. read (custfile, custrec);
2. while (not eof (custfile)) {printrec (custrec)
3. read (custfile, custrec);
```

Sample Of "Original" Code Fragment
Figure 2.1.2.1

2.1.2.1 Pretty Printers

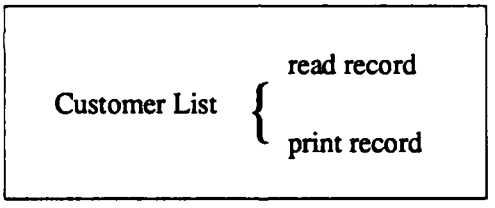
A Pretty Printer is a stylizer that addresses the placement of lines of code on a printed page or monitor. That is, it can ensure that indentation correctly represents the subordinate clauses, and that "begin"s and "end"s are lined up and occur in matching sets. Consequently, it represents static code structure at the detail level. It needs no specialized output devices, and was built to work with existing code. The output of a Pretty Printer is easy to understand, but it is difficult to envision how to extend a Pretty Printer's functionality. [Marti85c] See Figure 2.1.2.1.1 for an example of Pretty Printer output.

```
read (custfile, custrec);
while (not eof (custfile))
   {
      printrec (custrec);
      read (custfile, custrec);
   }
```

Example Of Pretty Printer Output
Figure 2.1.2.1.1

2.1.2.2 Warnier-Orr Diagrams

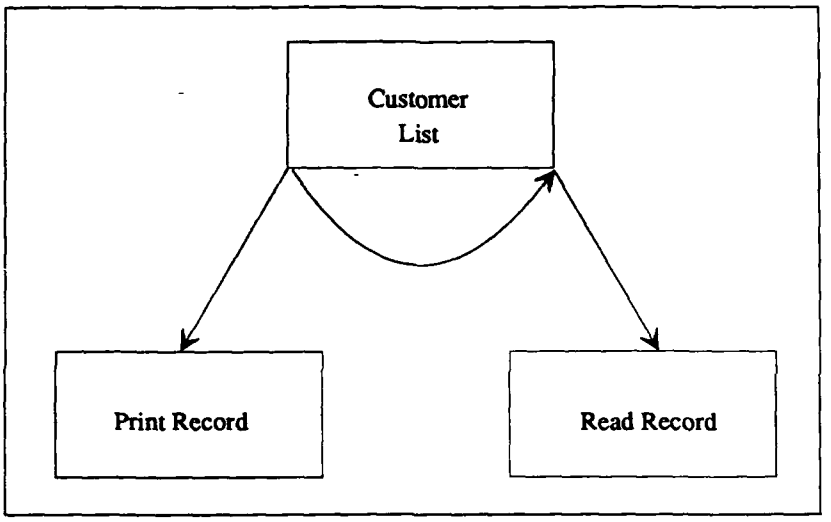
Warnier-Orr Diagrams are capable of depicting the structure of both code and data structures, both as an overview and in detail. They do not depict dataflow or program logic. Warnier-Orr Diagrams are displayed using braces that travel across a page, and are read from left to right. They need a graphics printer for hardcopy output, but it might be possible to display them using a non-graphics monitor. They work with code that was not designed using them, and are easy to learn. They do not lend themselves easily to the depiction of dataflow or program logic. [Marti85c] See Figure 2.1.2.2.1 for an example of a Warnier-Orr Diagram.



Example Of A Warnier-Orr Diagram
Figure 2.1.2.2.1

2.1.2.3 Jackson Diagrams

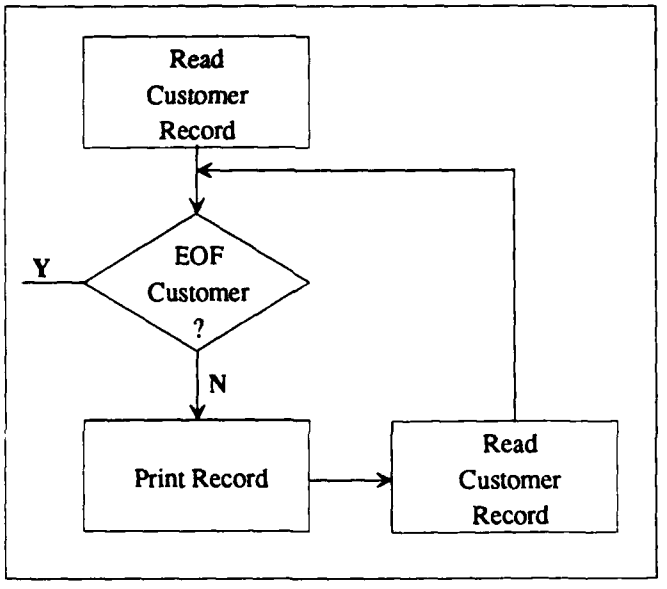
Jackson Diagrams depict the structure of code and data structures, but only as an overview. Detailed information about the code is not available, nor is program logic. Information is represented as a collection of boxes, collected into a tree structure, and is read from top to bottom. They require either a graphics printer or graphics monitor. They are useful on code not built with them, and are easy to learn. Extensions to their functionality seem reasonable, possibly by attaching dataflow information to the boxes in the tree. [Marti85c] See Figure 2.1.2.3.1 for an example of a Jackson Diagram.



Example Of A Jackson Diagram
Figure 2.1.2.3.1

2.1.2.4 Flowcharts

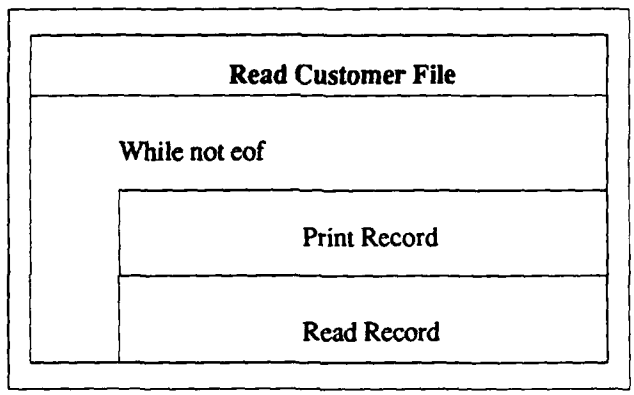
Flowcharts depict detailed information: the sequential representation of the code. They are not capable of depicting program structure. They are displayed as a collection of connected symbols, where the symbols have a meaning in and of themselves, e.g., a diamond shape represents a test in the code. To obtain hardcopy and electronic output, graphics capabilities are required. They can generate useful information about code not designed and developed with them, and are easy to learn. However, it is not easy to see how to extend their functionality to represent the program structure, or dataflow information. [*Marti85c*] See Figure 2.1.2.4.1 for an example of a Flowchart.



Example Of A Flowchart
Figure 2.1.2.4.1

2.1.2.5 Nassi-Shneidermann (NS) Diagrams

Nassi-Shneidermann (NS) Diagrams depict the static structure of a program, and the program logic, both at a detailed level. NS Diagrams are built from a small set of symbols for iteration, selection, and sequence. Complex code is represented by levels of nested symbols. They do not require graphics capabilities. However, using line-drawing graphics does aid in their appearance. NS Diagrams are useful on code that was not generated using them, and are easy to learn. While NS Diagrams were not designed to depict either dataflow or overview information, it is easy to see how to extend their functionality by adding symbols for tracking dataflow or providing overview information. It is not easy to envision a means for representing data structures using NS Diagrams. [Marti85c] See Figure 2.1.2.5.1 for an example of a NS Diagram.



Example Of A Nassi-Shneidermann (NS) Diagram
Figure 2.1.2.5.1

2.1.2.6 Pseudocode

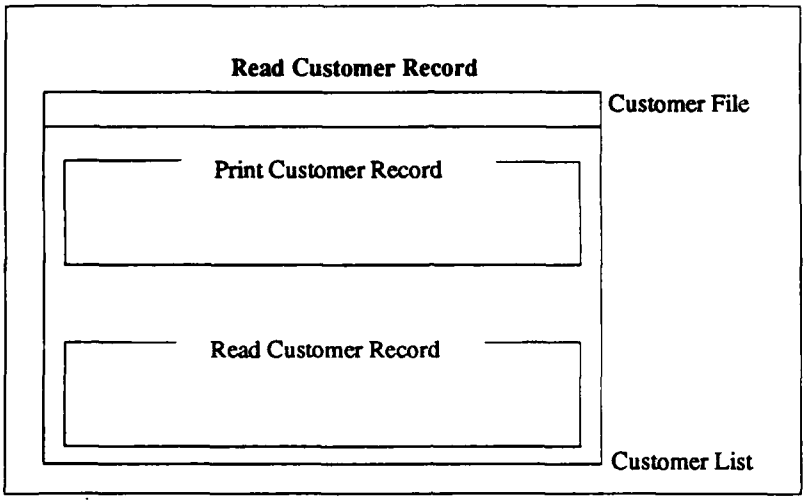
The generator of Pseudocode can define what information is included, and what is excluded. Pseudocode can range from a structured English language rendering of the program semantics to an essay-like discussion of the programmer's thoughts at the time of code generation. A Pseudocode generator needs no specialized output devices. It is possible to build Pseudocode from code not developed with it, but it can be difficult to achieve "good" Pseudocode. [Marti85c] See Figure 2.1.2.6.1 for an example of Pseudocode.

```
read a record
while record was read
   begin
      print the record
      read a record
   end
```

Example Of Pseudocode
Figure 2.1.2.6.1

2.1.2.7 Action Diagrams

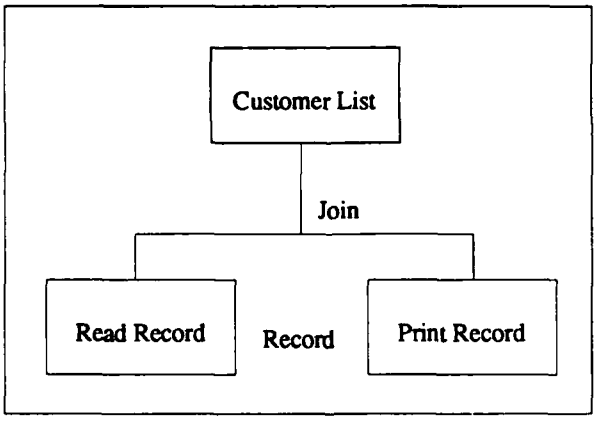
Action Diagrams can depict both an overview and a detailed representation of the code, and can be used to represent both program structure and dataflow. They are displayed by the use of various styles of brackets drawn around the code. They do not require graphics capabilities for display. They can be used with code not developed with them, and are considered easy to learn. [Marti85c] See Figure 2.1.2.7.1 for an example of an Action Diagram.



Example Of An Action Diagram
Figure 2.1.2.7.1

2.1.2.8 Higher Order Software (HOS) Charts

HOS charts can depict both details and an overview, but they were eliminated during the early stages of the survey due to their emphasis as a design tool. There are serious questions as to their usefulness with code not designed with them. They are considered very difficult to learn. [Marti85c] See Figure 2.1.2.8.1 for an example of a Higher Order Software (HOS) Chart.



Example Of A Higher Order Software (HOS) Chart
Figure 2.1.2.8.1

2.1.2.9 Cross-Reference/Usage Listing

Cross-References, or usage listings, depict detail in a user-defined manner. They may be used in some form of hypertext, i.e., interactively, or to simply track all the occurrences of a given identifier. A hypertext-oriented Cross-Reference generator would probably require a graphics monitor to be useful, while the canonical Cross-Reference generator requires no specialized output devices. Hypertext is difficult to learn, but extensions to a hypertext-based tool are limited only by the imagination of the developers. The canonical Cross-References are simple to understand, but difficult to envision extensions for. [Marti85c]

2.1.3 CONSTRAINTS APPLIED TO DOCUMENTATION

After examining the various documentation methodologies discussed above, it was apparent that certain constraints were relevant in order to maximize the desirability of a resulting tool to maintenance programmers. The most desirable documentation methodologies:

1. Depict both detailed and overview information, and both dataflow and code structure;

2. Do not require specialized output devices, either graphics printers or graphics monitors;

3. Generate useful information from code not developed with them;

4. Are perceived as being easy to learn; and

5. Are perceived to be easily extensible (if they do not make available all the information in constraint number 1).

It is possible, and probably obvious in hindsight, that the initial constraints would not survive for the duration of the project. However, decisions such as the limitation to line-printers-only forced the research to remain focused on the development of the documentation generation process rather than allowing the focus to get side-tracked on time-consuming issues such as graphics generation. It is clear that this restriction also made some of the documentation methodologies that were investigated appear less desirable than they might have been in a more graphics-oriented environment. By the time the final decision of documentation methodologies was made, the Macintosh was a viable system, and Postscript (and Postscript printers) was available. Consequently, graphical documentation was pursued after all. In spite of the power of these new technologies, the initial choices would probably be very similar today. However, those methodologies requiring more graphics support would not have been as easily rejected.

2.1.4 SELECTED DOCUMENTATION TYPES

The documentation types finally selected as most useful to maintenance programmers were NS Diagrams, Action Diagrams, and Cross-References. The consideration of extensions included the provision of a "subroutine header" symbol for NS Diagrams (containing

dataflow information), and the examination of hypertext toward an interactive Cross-Reference generator. Action Diagrams were not pursued because it was felt that NS charts would more effectively meet the familiarity-to-programmers criteria.

Thus, a subset of documentation methodologies available was examined, and reasonable constraints were applied to those methodologies to yield a small set of potential outputs for the tool under consideration. The more commonly used third generation languages were then surveyed to determine the inputs necessary for the maintenance tool.

2.2 EVALUATION OF TARGET LANGUAGE FAMILY

The research focused on the development of a methodology that would allow for the incorporation of several different languages into one common documentation system. This research differed from the work of others largely in that the target was to include several languages and forms of documentation, rather than a specific form of documentation for one specific language. Additionally, the documentation process was to be automatic, which required the sophisticated treatment of the source language, and the problems of semantics (different interpretation of similar constructs).

2.2.1 LANGUAGE REQUIREMENTS

Before beginning to compile the DL requirements that were a function of the requirements and capabilities of the individual target languages, the set of concepts common to all of the target languages was identified. The identification of common concepts allowed for the analogous treatment of those concepts, allowing concepts unique to each language to be given the individual consideration they required.

The target languages had differing levels of data definition requirements. FORTRAN, although in most cases the simplest, has some requirements that need special consideration. All of the languages allowed for variables to have single values. Also, all of the languages allowed for the vectors of these values, and the arbitrary vectors of vectors (arrays) of values. As mentioned, FORTRAN is generally thought to have the least requirements, but its handling of the "complex" type proved to require the inclusion of "complex" as a basic type in DL.

Many of the "modern" languages (such as C, Ada, and Pascal) allowed for the use of higher level types, including sets of values, ranges of values, and enumerated lists of values. In these cases, basic types are extended to provide for a more abstract representation of data than is typically allowed in FORTRAN. The structuring of data aggregates (records composed of several data fields) and its close relative, the union (or record variant), allows for the sophisticated grouping of data into an object upon which the program operates. Additionally, users may define their own types, according to their needs.

Aliases (multiple names for the same item), or equivalences, may be handled using variant records, where names "overlay" each other. Although this mechanism is difficult to read in its DL form, postprocessors are able to identify items that are aliases of each other using this approach.

Peculiar language requirements, not easily handled in a straightforward manner include source substitution (such as macros in C), statement functions, parameters in FORTRAN, and packaging and task management in Ada.

2.2.2 POTENTIAL PROBLEMS WITH SEMANTICS

When trying to identify the relationships between languages (and grammars), at least five areas exist with a high potential for causing problems. These five areas are:

- Differences in the semantics of syntactically similar code;
- Name spaces;
- Lexical versus dynamic scoping;
- Operator precedence; and
- Operator overloading.

Each of these areas provides significant power to programming languages, but requires ad hoc treatment. Additionally, conflicts between target languages result in contradictory requirements.

2.2.2.1 Differences in the Semantics of Syntactically Similar Code

The transformation of a programming language to documentation (or object code) requires the proper interpretation of input tokens to meaning. A typical language compiler provides this interpretation, called *semantics,* as the source is translated from readable tokens to machine instructions. In many cases, languages that are similar in appearance can vary significantly in the semantics (or meaning) applied to the source. For example, in Pascal, the statement

```
if i = 0 then ...
```

has a very different meaning from the C statement

```
if (i = 0) ...
```

in that the Pascal statement compares the value of 'i' to zero, and if equal, the 'then' statement is executed. On the other hand, this statement in C, although it appears similar, in fact assigns zero to 'i', and since the condition is always zero (the result of the assignment), the 'then' portion of the 'if' statement is never executed.

2.2.2.2 Name Spaces

The rules for names vary widely in the class of languages that were considered. The implementation variations for a single language such as FORTRAN made flexibility necessary. To provide the needed flexibility, the name space (organization of how items are uniquely identified by name) was constructed to allow for a "nested" specification, where

name uniqueness was required only within a localized group. The decision was made to have a postprocessor phase reduce (or flatten) the name space, and generate conflicts, if this was a characteristic of the particular implementation.

2.2.2.3 Lexical Versus Dynamic Scoping

Part of the name space problem is the accessibility of data due to the scoping requirements of the language. The *scope* of access is defined as the limits within the code that the name is *active*. In a block structured language, several different variables may all share the same name, yet be unique from each other. The immediate, or local, context defines which variable is being referred to. For example, Figure 2.2.2.3.1 presents a C code segment where each reference to an identifier has its scope defined in the commentary.

```
1   main()
2    {
3     int  i, j;
4     {
5      int   j;
6      i = 1;              // Refers to 'i' defined on line 3
7      {
8       int   i;
9       i = 2;             // Refers to 'i' defined on line 8
10      j = 3;             // Refers to 'j' defined on line 5
11     }
12     j = 5;              // Refers to 'j' defined on line 5
13     i = 6;              // Refers to 'i' defined on line 3
14    }
15    j = 7;               // Refers to 'j' defined on line 3
16    i = 8;               // Refers to 'i' defined on line 3
17   }
```

Scoping Example
Figure 2.2.2.3.1

2.2.2.4 Operator Precedence

Depending on the source language, and the operators defined in that language, the order in which the operations are performed may vary. Also, there are problems of consistency within the different implementations of the same language. For example, some FORTRAN compilers allow for expressions such as

```
PROD = SRC .AND. MASK
```

which performs a logical bit-wise "and" operation on 'SRC' and 'MASK'. Other FORTRAN compilers balk at such a construct. FORTRAN compilers that allow for the inclusion of the "and" operation in this manner have a different set of operator precedents than those that do not. The DL must provide for all operations so that the documentation generated represents the correct interpretation of the statement, regardless of which FORTRAN was used. (Note that different FORTRAN preprocessors may be necessary, and the research approach allows for this possibility.)

2.2.2.5 Operator Overloading

Although not perceived as something commonly used, operator overloading is relied upon by most programmers. For example, when expressing an equation in most programming languages, a programmer has learned to expect that an expression of the form of

$$a = b + c$$

is to be understood as 'a' is assigned the sum of 'b' and 'c'. In this discussion, consider only numerical interpretations of this equation. The instructions generated for this equation can vary widely, depending on the basic types of the variables 'a', 'b' and 'c'. If 'a' is of a basic type that is significantly different than the result of the sum operation, the compilation process must supply the conversions necessary to make the sum conform to the type of data that variable 'a' holds. For example, if 'a' is a "real" number variable, and the sum is a "fixed", or "integer" sum, then conversion from an "integer" representation to a "real" representation is needed. This process of conversion is known as *type coercion,* or *casting*.

This information, i.e., where type conversions are occurring, is useful as it may highlight an expression that may yield invalid results. For example, most computers can represent floating point (or "real") numbers that significantly exceed the valid range of fixed point (or "integer") numbers. Thus, if a floating point value is assigned to a fixed point variable, a range test may be in order.

With these ideas of what would be useful information to convey to the programmer, the next step was to bring all of these issues together in light of the technical proposal.

2.3 REEVALUATION OF TECHNICAL VALIDITY OF PROPOSED APPROACH

Having researched the documentation requirements, having designed data structures that allowed representation of the candidate methods, and having completed the definition of the DL, with its associated mappings to target source languages; a review of the proposed technical approach was performed. At that time, the emphasis was the consideration of how to implement the design. Since each language has a set of semantics specific to that language, and since it was necessary to avoid getting overly complicated with conflicting rules and contradictions between languages, the approach chosen was to divide the translation of source-to-documentation process into three distinct operations.

First, a language specific translator converts the source into the intermediate representation, or the DL. This is accomplished such that application of the DL semantics to the resulting DL source is equivalent to the original.

Next, the documentation language compiler translates the DL into common data structures, applying a standard semantic on all of the input. Semantics applied include such things as name binding and code grouping (partial block recognition). The result of the compilation process is a complete parse tree representing the input, with associated symbol tables.

Lastly, the documentation generators take the parse tree and symbol tables and create the documentation. Since the compiler knows nothing of which forms of documentation are ultimately to be generated, the parse tree and symbol tables contain all of the information available (and derived). Each documentation generator extracts what it needs from the parse tree and symbol table as it generates the documentation.

2.4 SUMMARY OF RESEARCH AND REEVALUATION

In summary, after surveying nine commonly used documentation methodologies, three were selected as the most appropriate for providing useful information for maintenance programmers. The survey of programming language concepts identified the areas of commonality and potential problems as well. The technical validity of the approach was examined, and was found to be acceptable in light of the research results.

At this point, the issues of documentation requirements were well understood. It was also clear that the three-phase, modular approach to the problem was a viable one. Consequently, the next step was to design the documentation language that would provide the intermediate representation of the concepts of interest.

CHAPTER 3 - DESIGN OF THE DL

The approach investigated in this research was to attempt to develop a "universal" documentation language having the following characteristics. It must be:

1. Sufficiently expressive to support the programming constructions of currently used programming languages;

2. Flexible enough to allow for exact representation of the semantics of several source languages in the intermediate representation; and

3. Complete. That is, it must be an intermediate representation which provides all the necessary information, yet which can be easily used by post-processors in generating the documentation.

The research began by reviewing commonly taught and used documentation techniques. As a constraining guide, documentation techniques were evaluated and "scored" according to their perceived usefulness in the software maintenance environment. From this survey, a couple of techniques (particularly suited to the software maintenance environment) were selected for further evaluation. The incorporation of documentation technique-specific requirements were investigated and added to the requirements of the documentation language.

Next, a survey of source languages: FORTRAN, Pascal, BASIC and C (with some consideration for Ada requirements) was conducted. From this class of languages, a set of representative control and data structures were derived, which allowed for an equivalent specification of the source languages in the derived language.

Using the derived language as an initial basis, the design of the documentation language became the focus of the research. Many of the details that had previously been glossed over now became items of significant effort. Included in these items were: type conversions (usually done implicitly in the source languages), the representation of constant expressions and their "folded" results, and the lexical scope (name space management) of identifiers and procedures.

As the definition of the documentation language proceeded, specification of the required data structures (representing the 'compiled' form of the documentation language) were developed. Several representative routines, written in source languages (FORTRAN, BASIC and C) were translated to documentation language, and then transformed into the data structure prototypes. From the resulting data structures, recreation (by hand) of the source language was accomplished, thus demonstrating the completeness of the transformation.

3.1 DOCUMENTATION REQUIREMENTS AFFECTING THE DL DEFINITION

Results of the survey of documentation techniques demonstrated that a significant level of detail was necessary for those techniques which proved most useful in a software maintenance environment. As a result of the survey, our predisposition toward "compiling" the source to an intermediate language appeared to be correct. Intermediate languages from which "code" could be executed provide not only the logic, but sufficient information for an analysis of the source code.

Although beyond the scope of the research, taking the approach of generating a full-blown intermediate language allowed for future enhancements such as building a code optimizer whose output would be optimizing recommendations which would allow for "tighter" code to be developed. For example, in many higher level languages, programmers are not encouraged to think in terms of how to express logic that is efficient, as the compiler attempts to perform common subexpression elimination, etc. Errors in the code are frequently generated when the user attempts to faithfully reproduce a common subexpression in many places, yet fails to do so. By having an "optimization recommendation" as a documentation output, programmers could be informed of where common subexpressions existed and alternative code could be suggested.

Other uses of the intermediate language approach would be to recognize and warn of expressions that appeared very similar, yet are different, under the presumption that these slight differences may in fact be errors. For example, a common subexpression may be used to select a particular element of an array, as in:

```
weap = tweap(cveh * 64 + cweap * 8 + cunit)
```

If the common subexpression "cveh * 64 + cweap * 8 + cunit" appeared in several places, it probably is an important difference to find a different subexpression such as "cveh * 63 + cweap * 8 + cunit", as an incorrect offset is probably going to be calculated. Note that it would still be up to the programmer to determine whether or not the highlighted expression was an error or not.

Still another future use of the intermediate language approach would be in the application of a heuristic to identify possible erroneous expressions. Simple heuristics applied to identifier, procedure and function names, for example, can reveal possible errors if there are names that are easy to misread. In languages like FORTRAN, where there are no "letter-case distinctions" in identifiers, or where identifiers are automatically truncated to some length (e.g., seven characters), it is easy to confuse identifiers such as:

Visually	Abbreviation	Truncation (at 7 chars)
MIN10	DOWRITE	DOREADRECORD
MINIO	DOWRIT	DOREADRESPONSE
MAX5		
MAXS		

A heuristic based on locating and displaying similar identifiers, according to several common rules, like the three above, can help locate problems.

Primarily, the driving force for selecting an intermediate language representation for the documentation language was the level of detail that many of the documentation techniques required. When examined with a view to future tools that could be included, as those suggested, this approach appeared to be worth the extra "front-end" effort.

3.2 TARGET LANGUAGE REQUIREMENTS AFFECTING THE DL DEFINITION

Central to the approach was the development of a single universal documentation language. This language, by this requirement, must have the properties of all supported languages in that no construction in a target language would be unrepresentable in the documentation language. As such, a program (represented in documentation language) must be as complete (although not necessarily executable) as it was in the source language. Given the large number of existing languages (both general and special purpose), some focusing and elimination of languages was necessary.

3.2.1 RESTRICTIONS

The candidate languages were required to be general purpose, third generation languages. The candidate languages must include such things as:

1. Enjoy popular "industry standard" acceptance. Since a large body of candidate code, where automatic documentation would seem to be very useful, exists, esoteric languages were not given consideration;

2. Be useful for writing almost any type of program, not being restricted to a particular class of problems. For example, languages that solve equations, or other domain specific problems were not considered;

3. Procedural languages. Most of the newer fourth generation languages enhance the ease of programming by doing useful things without detailed procedural specification. An advantage to using fourth generation techniques is that, generally, there is a clear distinction between abstract and implementation details, thus allowing high level design information to survive the implementation phase which allows design-level (abstract) to remain separated from the detail level code;

4. Static binding. One common characteristic of most third generation languages is that static evaluation of source is able to reveal the logic of the program. Our research eliminated any languages that provided (in the language) for dynamic binding (selection of the routine to execute at run-time); and

5. Overloading of identifiers. Ada allows for overloading of identifiers. For example, more than one variable named 'i' may exist, provided the definitions of both variables have different types. Selection of which 'i' is being referenced in the program is done by the compiler. Which 'i' is selected is determined by the

program context or usage of 'i'. Other languages allow similar overloading (such as C++ and Modula), but these features were not required for the primary languages of interest (FORTRAN, Pascal, and C).

For the "proof of concept" model, further restrictions were applied to increase the likelihood of completing an initial implementation of the model. These included:

1. Depend on a language specific preprocessor to translate the source language into documentation language;

2. Select only languages where the straightforward translation of the original source was possible. In many respects, an exception to this rule was FORTRAN. For example, to allow FORTRAN to remain in the candidate set of languages (simply because of its popularity), documentation of FORTRAN input/output was eliminated from the proof of concept implementation; and

3. Languages that did not present any unique problems, even though they enjoy a large share of usage, were eliminated.

As a result of applying these restrictions, the initial source languages FORTRAN and C were selected. Since the C language served as the basis for the documentation language specification, a minimal amount of preprocessing was necessary, thus making it the initial source language.

3.2.2 RETENTION OF ALL INFORMATION

Having used the intermediate language generation approach, it was decided to retain all of the original logic in tuples of operator with multiple operands, organized in a parse tree. The intermediate language was designed in such a manner that, if necessary, executable code could be generated from the compiled code (or parse tree). Additionally, transformations from source language input, to the parse tree, to source language were shown to create equivalent (but not necessarily identical) output. Differences between input and output were in the usage of "white" space such as spaces, tabs and comments.

Constant expressions, in addition to being reduced to values, were retained as intermediate language expressions thus allowing the usage of constant expressions in data structure declarations (array size definition), yet retaining access to the constituent parts used in defining those "calculated" constants.

3.2.3 EQUIVALENT SPECIFICATION

It was important that source statements be expressible in the documentation language so that accurate documentation of the actual code could be done. In FORTRAN, however, where input/output is "built-in" to the language, some of the more complex interactions between the FORMAT statement and the argument list proved too much for the prototype implementation. As a simplification, specification of built-in input/output was not considered. More research is needed in integrating the "run-time" dynamic nature of FORTRAN FORMAT statements with an otherwise static evaluation of source code.

Data structures were implemented in the documentation language, to support user data structure descriptions. Support of the FORTRAN EQUIVALENCE, for example, is handled by requiring the FORTRAN preprocessor to express the equivalenced variables by specifying variant structures (records) in a union in which each structure overlays the companion structures in the union.

Although each language tends to have a unique syntax for representing logic and control flow, a significant amount of commonality exists. The classes of flow control statements reduce to sequential statements and expressions, decision statements, repetition or looping statements, selection statements, and subprogram or function execution statements. Control structures in the documentation language needed to allow for all forms required by the selected class of languages.

By mapping, by hand, the variants of statements allowed by the source languages, the documentation language was developed, and confirmed to have the capacity to represent the source language. It was found that the selected base language (X3J11 C) was capable of supporting all of the data and control structures needed by the source languages.

3.3 DEFINITION OF THE DL

As stated, X3J11 C provided the base for the documentation language, DL. Support of all of the documentation language requirements, however, required several deviations from that starting point.

One area of extension was in constant and constant expression representations. Documentation of data structures and their usage required that constant expressions be evaluated in order to provide correct mapping of the structure/union definitions used in FORTRAN EQUIVALENCE processing, and unions in general. For example, if a structure contains an array, which has a constant expression specifying the number of elements, it is necessary to state the storage requirements of that array. If the constant expression involves several constants (symbols having values such as MAXRECS, or numbers such as 10), recording both the expression and reducing that expression to a value used in storage allocation calculations was required.

Another area that required extension to the X3J11 C model was support of nested procedure definitions, as allowed in Pascal and Ada. A more sophisticated symbol table than required by C was necessary to allow for lexical scope binding in those cases.

An area required for Ada, but not fully implemented in the documentation language was overloading of identifiers. In order to allow for future inclusion of Ada as a source language, the specification of the symbol table routines included structures allowing for name-type binding rather than name-only binding. In this way, the compiler design was such that first a name-type binding was attempted, and if that failed, name-only binding was attempted. This design allowed the compiler to more easily recognize type conversions, and warn about them.

The inclusion of FORTRAN required in the base language the implemention of type complex. This required expression evaluation to overload the arithmetic symbols '+', '-', '*' and '/' in addition to assignment '=', and the respective augmented assignment '+=', '-=', '*=', and '/='. Some of the augmented assignments needed to be properly diagnosed as invalid when using the complex type, such as '^=', '|=' and '&='.

Extensions to the base X3J11 C language were resisted at every point in order to really justify the extension. This resulted in a language relatively free of non-standard or quirky specifications. It was interesting to note how far the X3J11 C specification went in satisfying the requirements of DL, the documentation language.

For example, most compilers implement "constant folding" at a very early point in the translation of the program. As a result, individual constants are reduced to values. This "folding" tends to obscure, or hide (in many applications) the constants used in values computed during the compilation, and their influence on other constants or constant expressions used system-wide. To the user of automatically generated documentation, it is generally useful to know both the "value" of the constant expression, and the constituent constants used in creating that value.

3.4 VERIFICATION OF DEFINITION FOR DESIGN COMPLETENESS

Concomitant with the definition of DL, data structure prototypes were hand drawn to represent the data structures that would be implemented. In addition, symbol table management routines were specified that provided a minimal interface and yet were sufficient to build the parse tree.

In addition to generating the symbol table structures, the structures were exercised for completeness by parsing several routines, representing the full complement of structures provided by the candidate languages. Samples of documentation language were provided for all candidate language control structures to provide both documentation of how the source structures mapped to prototype structures, and to exercise the symbol table structures.

Completeness was confirmed by traversing the resulting parse tree (again by hand) and generating the source (using an in-order traversal). Enough effort and samples were done to confirm that the proposed language mapped to specific structures, and that those structures faithfully represented the proposed language.

3.5 SUMMARY OF DL DESIGN AND EVALUATION

Although the documentation language was initially based on the proposed X3J11 C programming language, it was found that surprisingly few enhancements to that base were necessary in order to retain the information that was "interesting" to those using the resulting documentation.

CHAPTER 4 - IMPLEMENTATION OF THE DL DESIGN: THE RESEARCH PROTOTYPE

The implementation of the approach began with the DL compiler. Once the compiler was considered fairly stable, work was initiated on pre- and postprocessors for it. Each of these entities is discussed in this chapter, along with examples of prototype operation, and two unexpected results of prototype development.

4.1 DOCUMENTATION LANGUAGE COMPILER

Using the data structure diagrams as a guide, the compiler was implemented to build the parse tree and symbol table that represented the input DL. Work on the compiler was essentially complete at the time of our Interim Report. For details regarding this effort, the reader is referred to that report (Documentation in a Software Maintenance Environment, DTIC reference number AD A185 892). The documentation language grammar was defined in a modified BNF which was accepted by the parser generator yacc. Additionally, the lexical analysis was specified using lex, which generates an FSM (Finite State Machine) that recognizes language tokens from the input. (Both yacc and lex are UNIX utilities.)

4.2 PREPROCESSOR

Work was begun on a FORTRAN to DL preprocessor, but due to other considerations, did not progress beyond the design stage. Several interesting results arose during that work which bear discussion. Examples are provided, demonstrating the expected results of translating FORTRAN to DL. Concepts discussed are:

- Data types;
- Expressions and statements;
- Control flow;
- Declarations / Common blocks / Equivalences; and
- Input / Output.

Problems and items of interest are pointed out in the relevant sections. The discussion is not meant to be exhaustive, but rather to highlight some of the more interesting concepts.

4.2.1 DATA TYPES

Figure 4.2.1.1 depicts the anticipated mapping of FORTRAN data types to the DL.

FORTRAN	to	DL
ARRAY (I,J,K) (simple)		array [k] [j] [i]
BYTE		char
CHARACTER		char [2]
CHARACTER*n		char [n+1]
COMPLEX		complex
INTEGER		long
INTEGER*2		short
INTEGER*4		long
LOGICAL		char
REAL		float
REAL*4		float

Mapping Data Types From FORTRAN To The DL
Figure 4.2.1.1

In order to represent FORTRAN characters in the DL, one extra byte must be added because the DL signals the end of a character variable with a null terminator, just as C does.

4.2.2 EXPRESSIONS / STATEMENTS

Looking at the conditional expression in the statement

```
IF (WAVE1 .GT. 3188. .AND. WAVE1 .LT. 3195.) KWAVE = 7
```

the parameter WAVE1 was passed by reference (standard FORTRAN). In order to achieve the same functionality in the DL, the pointer must be explicitly specified:

```
if ((*wave1 > 3188.0) && (*wave1 < 3195.0)) kwave = 7;
```

4.2.3 CONTROL FLOW

FORTRAN control structures include the DO loop, the block IF (in the newer versions), the logical IF, the arithmetic IF, and the GOTO.

In the FORTRAN block segment in Figure 4.2.3.1, lines 8 through 20 effect a bottom-exit loop. The same code is depicted, in the DL, in Figure 4.2.3.2, with a brute force rendering of the loop. A more elegant rendering is found in Figure 4.2.3.3, replacing the FORTRAN label/goto combination with a DL do-while loop.

(Throughout the rest of this discussion, the line numbers on the far left in code segments are for the ease of identification in the text; they are not part of the original FORTRAN or DL code segments.)

```
1    IF ( IR .NE. IRPH ) GOTO 999
2    IF ( PHASE .LT. 0.1 .OR. PHASE .GT. 2.0 ) GOTO 999
3    IF ( UNI .GT. 0.1 ) INPT  = IFIX( UNI )
4    IF ( UNO .GT. 0.1 ) IOUT  = IFIX( UNO )
5    IF ( UNC .GT. 0.1 ) ISECND= IFIX( UNC )
6    IF ( UNF .GT. 0.1 ) IHISTU= IFIX( UNF )
7    IPHAS = IFIX( PHASE + .001 )
8   5 CONTINUE
9    IOLD = IPHAS
10    WRITE(IOUT,300) IPHAS
11  300 FORMAT(1H1///58X,17(1H*)/58X,1H*,15X,1H*/58X,
12   *17H*   COMBIC    */58X,11H*   PHASE ,I1,5H   */58X,1H*,
13   *15X,1H*/58X,17(1H*))
14    IF ( ECHO .GT. 0. ) WRITE ( IOUT, 200 ) IR, PHASE, UNI,
15   * UNO, UNC, UNF, ORDRS, ECHO
16    IF (IPHAS .EQ. 2) CALL DSPH2( ICLMAT, IPHAS, KWAVE, STRANS,
17   * ORDRS, IERR, ECHO)
18    IF ( IOLD .EQ. IPHAS ) GOTO 555
19    IF ( IERR .GT. 0 ) GOTO 555
20    IF ( IPHAS .EQ. 1 .OR. IPHAS .EQ. 2 ) GOTO 5
21    GOTO 555
22  999 CONTINUE
```

FORTRAN Control Flow Example

Figure 4.2.3.1

```
1   if ( !(ir != irph) )
    {
2     if ((( phase < 0.1) || (phase > 2.0) )
      {
3    if ((uni > 0.1)) ioun.combic.inpt = ifix(&uni);
4    if ((uno > 0.1)) ioun.combic.iout = ifix(&uno);
5    if ((unc > 0.1)) ioun.combic.isecnd = ifix(&unc);
6    if ((unf > 0.1)) ioun.combic.ihistu = ifix(&unf);
7    iphas = ifix(&phase + &0.01);
8  L5:
9    iold = iphas;
10   /* write */
...
14   if ((echo > 0 ))  /* write */
...
16   if ((iphas == 2)) dsph2(iclmat, &iphas, &kwave, strans,
17                           &ordrs, ierr, &echo);
18   if ((iold == iphas)) goto L555;
19   if ((*ierr == iphas)) goto L555;
20   if ((iphas == 1) || (iphas == 2 )) goto L5;
21   goto L555;
22 L999:
```

The DL Rendering Of Figure 4.2.3.1

Figure 4.2.3.2

```
8   do {
        <lines 9 through 19>
20  while ( !((iphas == 1) || (iphas == 2)));
```

Better DL Rendering Of Loop Structure

Figure 4.2.3.3

Note that the sense of the overall test must be reversed from line 20 (4.2.3.1) to line 20 (4.2.3.2), in order to achieve the same functionality. The same reversal took place in translating line 1 from FORTRAN to DL.

The reference to IFIX in line 3 (4.2.3.1) is a function call, so to achieve FORTRAN's pass-by-reference for UNI, Figure 4.2.3.2 shows &uni. The translation of the WRITE statements in lines 10 and 14, and the FORMAT statement associated with line 10 (all 4.2.3.1) are discussed in section 4.2.5.

Another variable reference which has changed appearance is INPT in line 3 (4.2.3.1), becoming ioun.combic.inpt in Figure 4.2.3.2. INPT is a member of a COMMON block, and this transition is described in section 4.2.4.

4.2.4 DECLARATIONS / COMMON BLOCKS / EQUIVALENCES

The FORTRAN DATA statement, which serves to initialize variables, translates directly across to DL, in that

```
DATA PASQC/ 'A', 'B', 'C', 'D', 'E', 'F'/
```

becomes

```
char pasqc[7] = 'ABCDEF';
```

in the declaration section of the translated code.

The PARAMETER statement, declaring and initializing symbolic constants, becomes a macro definition in the DL:

```
  PARAMETER (
* PI = 3.14159625,
* TWOPI = 2.0 * PI )
```

becomes

```
#define  PI        3.14159625
#define  TWOPI     2.0 * PI
```

where the symbolic names PI and TWOPI retain their case in translation as self-documenting features.

The IMPLICIT statement, restricting the types of identifiers, has no analogue in the DL. It would be consumed and incorporated into the set of control information that drives the translation process.

The FORTRAN COMMON statement, or block, serves to provide the functionality of global variables. The COMMON block maps cleanly to the DL union as follows: Given that the statement

```
COMMON /IOUN/ INPT, IOUT, ISECND, IHISTU, MAXVAL, MAXREC, INIT
```

appears in two subroutines: COMBIC and SDREAD, Figure 4.2.4.1 contains the DL rendering of the common block.

```
union {
  struct {
    long inpt,
         iout,
         isecnd,
         ihistu,
         maxval,
         maxrec,
         init;
  } combic;
  struct {
    long inpt,
         iout,
         isecnd,
         ihistu,
         maxval,
         maxrec,
         init;
  } sdread;
} ioun;
```

FORTRAN To DL: Common Blocks
Figure 4.2.4.1

Following the DL access requirements, a reference to IOUT in subroutine COMBIC in the FORTRAN code would become ioun.combic.iout in the DL version.

The use of the union construct also allows for the treatment of COMMON blocks which have different names, or types, for some, or all, of their elements. The reader may note that the dangers associated with mixing and matching types in a COMMON block also get transferred to the DL representation.

EQUIVALENCES are treated in an analogous manner to achieve the "tagless" variant record concept common to Pascal.

4.2.5 INPUT / OUTPUT

Input/output can be considered one of the most powerful of all FORTRAN constructs. The associated FORMAT statements provide formatting control information for both input and output. The WRITE and (simple) FORMAT statements contained in Figure 4.2.5.1 are combined in the DL to yield one fprintf, contained in Figure 4.2.5.2, where the underscore, "_", depicts blank spaces.

```
      WRITE (IOUT, 300) IPHAS
  300 FORMAT(1H1///58X, 17(1H*)/58X, 1H*, 15X, 1H*/58X,
     *17H* COMBIC */58X, 11H* PHASE, I1, 5H */58X, 1H*,  *15X,
     *1H*/58X, 17(1H*))
```

FORTRAN Input / Output Example
Figure 4.2.5.1

```
   fprintf( iout, ^L, \n, \n, \n,
"                                                                    ",
"*****************", \n,
"                                                                    ",
"*                 *", \n,
"                                                                    ",
"*____COMBIC___*", \n,
"                                                                    ",
"*____PHASE_%d___*", iphas, \n,
"                                                                    ",
"*                 *", \n,
"                                                                    ",
"*****************" );
```

The DL Version Of Figure 4.2.5.1
Figure 4.2.5.2

Execution of the fprintf in Figure 4.2.5.2 yields the output depicted by Figure 4.2.5.3. (The "x" after PHASE is the single-space integer specified by I1, and the number of blank spaces has been reduced to 29, or half of the original 58.):

```
----------<top of page>

                              *****************/
                              *                 */
                              *____COMBIC___*/
                              *                 */
                              *____PHASE_x___*/
                              *                 */
                              *****************/
```

Output Generated From Figure 4.2.5.2
Figure 4.2.5.3

4.3 POSTPROCESSOR

Next, the Nassi-Shneidermann Diagrammer was implemented. This design was partitioned to allow for a modular approach. Consequently, a Nassi-Shneidermann Diagramming Language (NSL) was specified. The compiler was modified to generate the NSL from the parse tree. A second module was written that translated the NSL to printer output, so it was now possible to go from a DL representation of a description to NS Diagrams.

4.3.1 NSL SPECIFICATION

NS Diagrams, in their standard form, were used as the starting point. [Marti85c] Extensions to the standard NS Diagrams were incorporated to include information such as routine headers and variable declarations, including name, type, and off-page connectors. To simplify the process of creating NS Diagrams, a simplified language was developed, which is called NSL.

Statements in NSL look similar to those statements available in many other languages. There are constructs that allow for compound statements and iteration statements (for, while, until and loop). Control statements were also added to allow for the control of the documentation

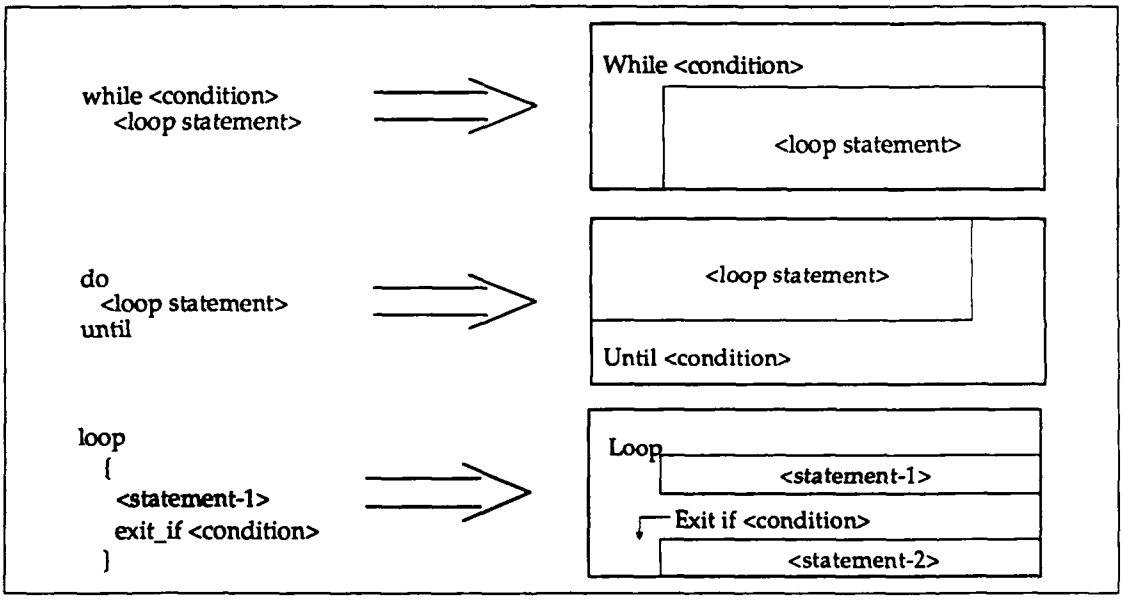
layout (page, page-break, routine header, and off-page connectors). Several examples of "real code" were generated to verify that NSL was complete for the set of operations created by the compiler, in the parse tree.

4.3.2 NSL STATEMENTS TO NS SYMBOLS

Concomitant with the specification of NSL was the form of the pictorial representation of the NSL statement. Four classes of NSL statements were developed, which are Iteration, Selection, Control, and Other.

4.3.2.1 Iteration Statements

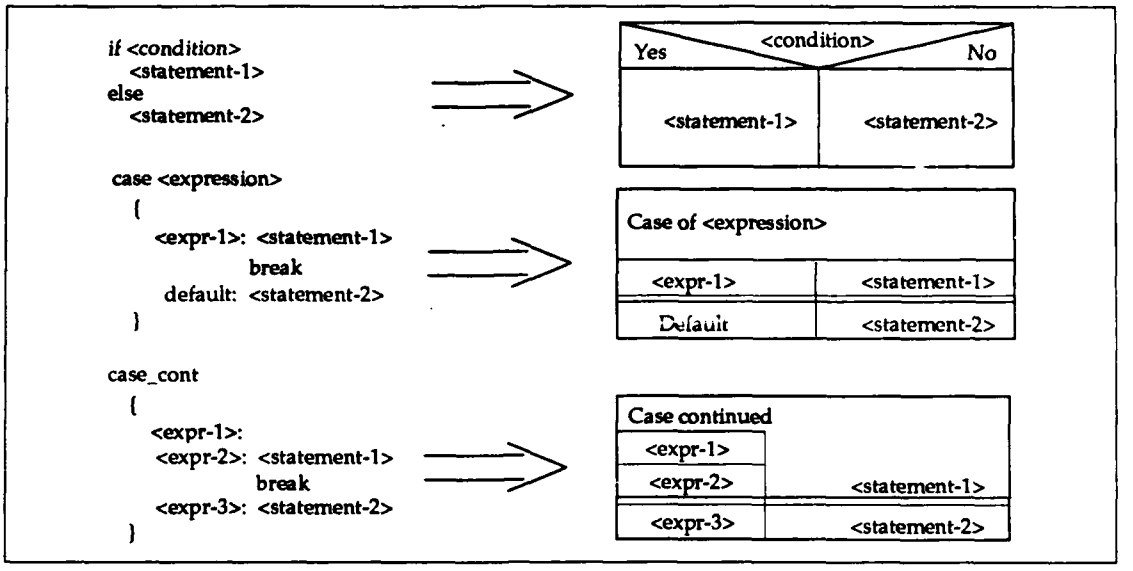
Looping, or iteration statements, are used to represent those programming constructs that control the repeated execution of a statement or group of statements, based on an arbitrary condition. Loops are represented by 'while', 'until', and 'loop' structures. See Figure 4.3.2.1.1 below, for examples of iteration statements.



DL To NSL: Iteration
Figure 4.3.2.1.1

4.3.2.2 Selection Statements

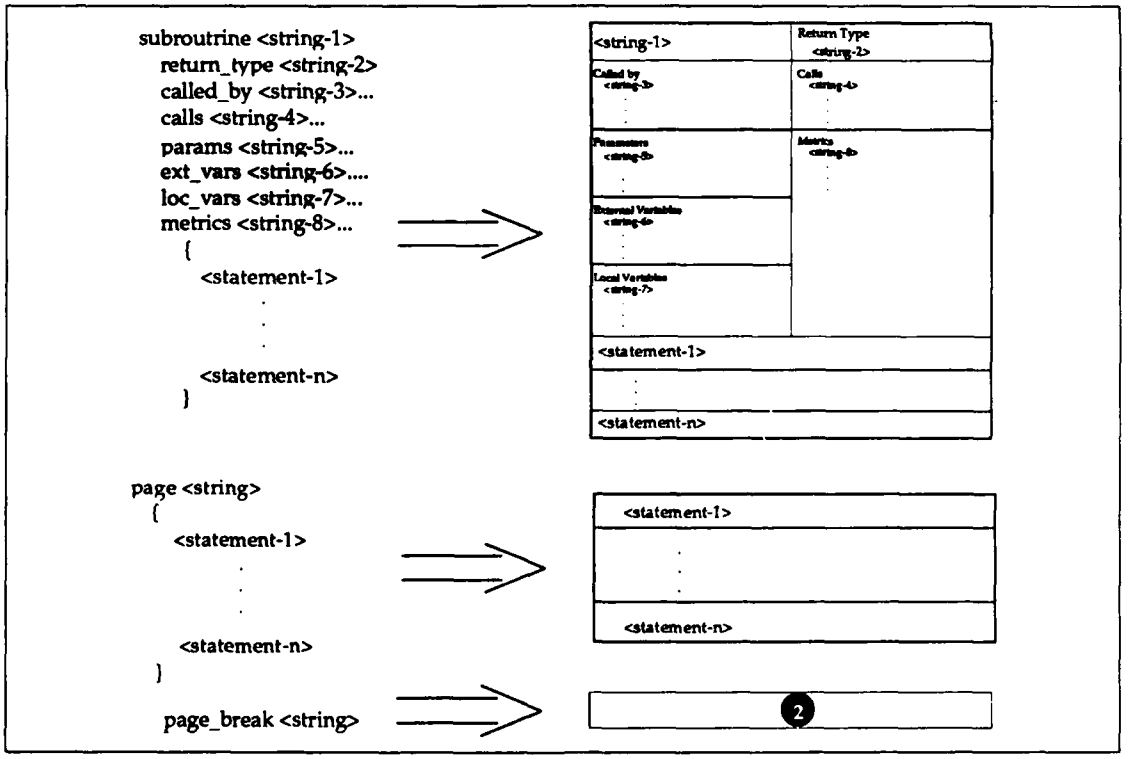
The conditional flow of control in the program logic is represented in one of two forms. These are the 'if' and 'case' statements. In some logic, there are more 'cases' generated than will fit on a single page or the same page as where the 'case' starts. To represent a long case, the 'case_cont' structure was added to specify 'case continuation'. See Figure 4.3.2.2.1 below, for examples of selection statements.



DL To NSL: Selection
Figure 4.3.2.2.1

4.3.2.3 Control Statements

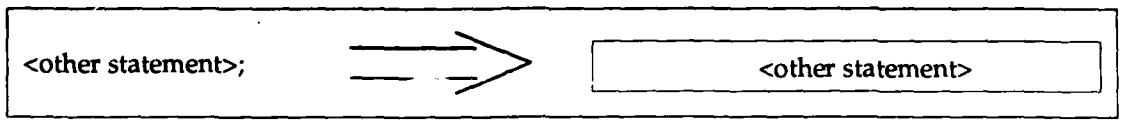
Pagination and labeling is specified using the control statements. A simplification of the design was accomplished by making it the responsibility of the NSL generator (whether done manually or automatically) to keep track of how much information is on the current page. See Figure 4.3.2.3.1 below, for examples of control statements.



DL To NSL: Control
Figure 4.3.2.3.1

4.3.2.4 Other Statements

All statements not included in the Iteration, Selection, and Control statements are represented in the class Other. Common expressions and assignments fall into this class of statements. See Figure 4.3.2.4.1 below, for an example of an other statement.



DL To NSL: Other
Figure 4.3.2.4.1

4.3.3 NSL GENERATION

The generation of NSL from the parse tree proved to be a straightforward process. The information in the subroutine header has proven to be a substantial improvement to the standard NS Diagramming technique because in the standard form, no information is readily maintained nor is it represented regarding the scope of the variables accessed in the rest of the Diagram. This information (type and scope of variables) was readily available in the parse tree, to include whether or not that variable is accessed or modified. As previously stated, the design of the NSL interpreter placed the responsibility of what appears on the page on the generator of the NSL.

Many of the additional control statements (page, etc.) were derived as a result of making groups of statements fit on a page in a reasonable manner. Several pagination strategies were explored in order to develop a good NS Diagram. The final strategy was to make every attempt possible to keep the main logic of an entire routine on the same page. If the routine was too long to fit, then blocks of sequential statements were moved to another page (using an off-page connector), and if loops were present, the generator attempted to keep the loop on the same page. It should be noted that a loop, regardless of its length, can typically be represented in two lines ('exit' loops excluded) by using an off-page connector as the body of the loop for one line and the condition as the other line.

4.3.4 NSL INTERPRETER

The NSL representation of the source code was interpreted to generate a Postscript description of that source. That Postscript representation was then processed by a Postscript engine on a laser printer, yielding the final NS Diagram.

4.4 AN EXAMPLE OF PROTOTYPE OPERATION

Figures 4.4.1 through 4.4.5 depict a C code segment being traced throughout the prototype operation, yielding the DL representation, the associated symbol table dump, the routine in the NSL representation, and the final NS Diagram.

```
long factorial (i)
  long i;
{
 if (!i)
   return 1;
 else return (i * factorial (i-1));
}

main ()
{
 printf("the factorial of 10 is %ld\n", factorial (10));
}
```

The C-Coded Routine
Figure 4.4.1

```
printf(char*, ...);

function long factorial (i)
  long i;
{
 if (!i)
   return 1;
 else return (i * factorial (i-1));
}

function main ()
{
 printf("the factorial of 10 is %ld\n", factorial (10));
}
```

The DL Representation
Figure 4.4.2

```
Oct 21 10:13 1988  fr_test.sym Page 1

Parsing input
   2:   3:   4:   5:   6:   7:   8:   9:  10:  11:  12:  13:  14:  15:  16:
  17:  18:  19:  20:
Dumping all internal table information

Block Index
  Id# Level Name
  --- ----- ----------
    0     0 anchor
    1     1   external
    2     2     printf
    3     2     factorial
    4     2     main


Block # 0  anchor              (Level:  0)
  bl_type:   BT_BASE
  bl_return: <no return type>
  bl_parent: <no parent>
  bl_formal: <none>

TBL_TYPE   (used 18/25 entries) of DT_TYPE
Note: '!' after Size - full access requires far pointer.
 Id#  Name              Class       Size        References
----- ----------------- ----------- ----  ----------------
 0/ 0  complex           TY_COMPLEX     8
 0/ 1  default           TY_ALIAS       2
         of TBL_TYPE  0/11
 0/ 2  default const     TY_CONST       2        Value: 0
 0/ 3  double            TY_DBL         8
 0/ 4  double complex    TY_DCMPLEX    16
 0/ 5  enumerator        TY_ENMRTR      4
 0/ 6  float             TY_FLT         4
 0/ 7  prototype far     TY_FAR         4
 0/ 8  prototype near    TY_NEAR        4
 0/ 9  signed char       TY_CHAR        1
 0/10  signed long       TY_LONG        4
 0/11  signed short      TY_SHORT       2
 0/12  unsigned char     TY_CHAR        1
 0/13  unsigned long     TY_LONG        4
 0/14  unsigned short    TY_SHORT       2
 0/15  void              TY_VOID        0
 0/16 int                TY_ALIAS       2
         of TBL_TYPE  0/ 1
 0/17 unsigned           TY_ALIAS       2
         of TBL_TYPE  0/14

TBL_BLOCK  (used 1/1 entries) of DT_BLOCK
 Id # Blk # Name
----- ----- ----------
 0/ 0     1  external


Block # 1  external            (Level:  1)
  bl_type:   BT_BLOCK
  bl_return: <no return type>
```

```
Oct 21 10:13 1988  fr_test.sym Page 2

  bl_parent: anchor
  bl_formal: <none>

TBL_TYPE   (used 3/9 entries) of DT_TYPE
Note: '!' after Size - full access requires
 Id#  Name              Class       Size
----- ----------------- ----------- ----
 1/ 0  () returning      TY_FUNC        2
 1/ 1  () returning      TY_FUNC        4
 1/ 2  near pointer      TY_NEAR        4     t

TBL_VAR    (used 3/9 entries) of DT_SYMBOL
 Id#  Name              Usage       Type      At
----- ----------------- ----------- -----     --
 1/ 0 factorial         US_DECL     1/ 1
 1/ 1 main              US_DECL     1/ 0
 1/ 2 printf            US_DECL     1/ 0

TBL_BLOCK  (used 3/9 entries) of DT_BLOCK
 Id # Blk # Name
----- ----- ----------
 1/ 0     2  printf
 1/ 1     3  factorial
 1/ 2     4  main


Block # 2  printf              (Level:  2)
  bl_type:   BT_PROTO
  bl_return:   default
  bl_parent: external
  bl_formal:   TBL_VAR  2/ 0 TBL_VAR  2/ 1

TBL_VAR    (used 2/9 entries) of DT_SYMBOL
 Id#  Name              Usage       Type      At
----- ----------------- ----------- -----     --
 2/ 0 ?0                US_FPARM    1/ 2
 2/ 1 ?1                US_FPARM    0/10


Block # 3  factorial           (Level:  2)
  bl_type:   BT_FUNC
  bl_return:   signed long
  bl_parent: external
  bl_formal:   TBL_VAR  3/ 0

TBL_CONST  (used 1/1 entries) of DT_TYPE
Note: '!' after Size - full access requires
 Id#  Name              Class       Size
----- ----------------- ----------- ----
 3/ 0  long constant     TY_CONST       4

TBL_VAR    (used 1/1 entries) of DT_SYMBOL
 Id#  Name              Usage       Type      A
----- ----------------- ----------- -----     --
 3/ 0 i                 US_FPARM    0/10
```

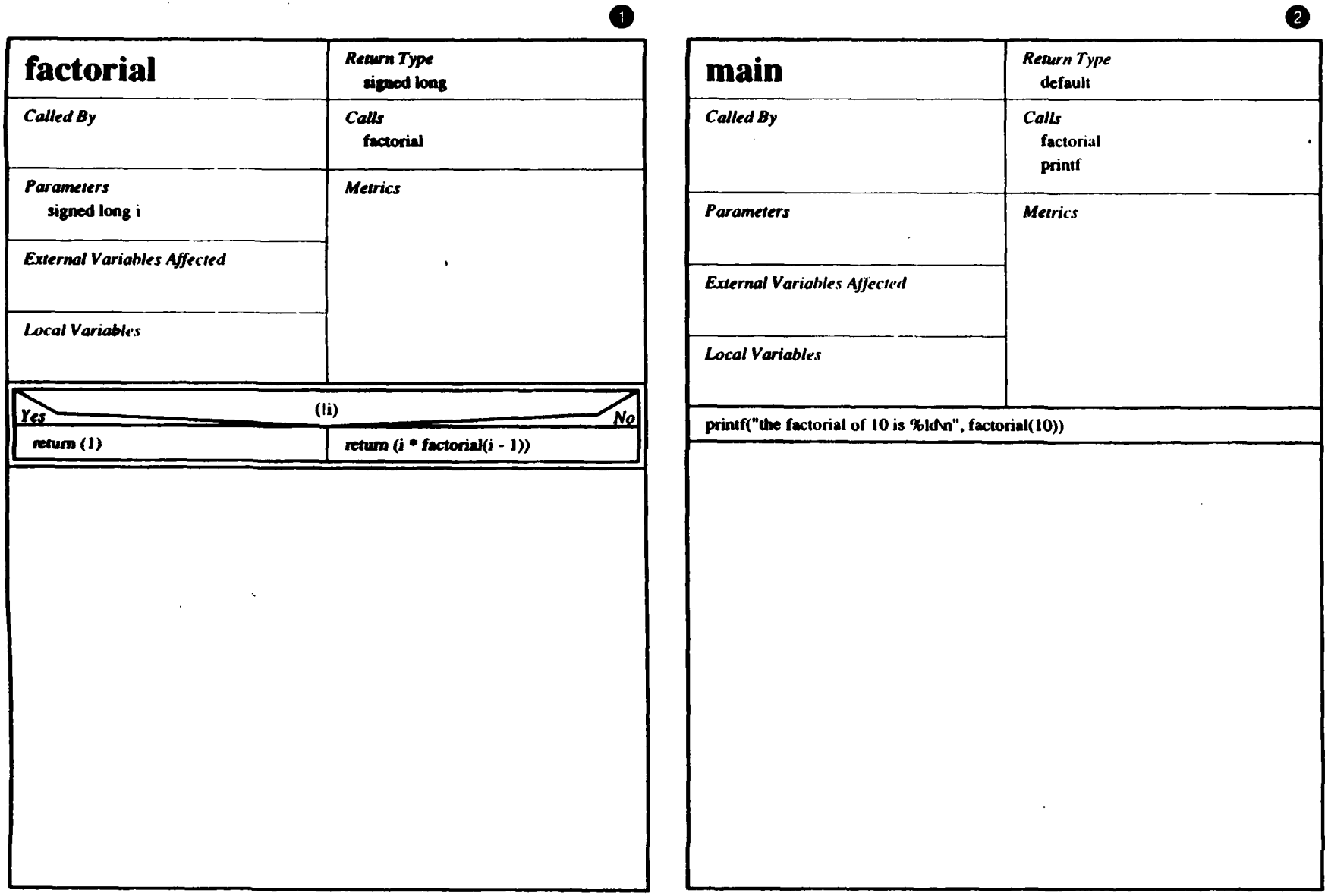
The Symbol-Table Dump
Figure 4.4.3

```
Page 2

of OT_TYPE
cess requires far pointer.
   Size        References
--- ----  ----------------
      2      TBL_TYPE  0/ 1 ()
      4      TBL_TYPE  0/10 ()
      4   to TBL_TYPE  0/ 9 near/volatile

of OT_SYMBOL
   Type     Attributes/References
   ----     ---------------------
   1/ 1
   1/ 0
   1/ 0

of OT_BLOCK




evel:  2)


_VAR  2/ 1

of OT_SYMBOL
   Type     Attributes/References
   ----     ---------------------
   1/ 2
   0/10


evel:  2)



of OT_TYPE
cess requires far pointer.
   Size        References
--- ----  ----------------
      4      Value: 1

of OT_SYMBOL
   Type     Attributes/References
   ----     ---------------------
   0/10
```

```
Oct 21 10 13 1988  fr_test sym Page 3

TBL_CODE   (used 2/9 entries) of OT_TYPE
 Id0  Quadruple Sequence
---- ------------------
 3/ 0     001100000001 TBL_QUAD  3/ 2

 3/ 1     003100000000 TBL_QUAD  3/ 4 TBL_QUAD  3/ 5


TBL_QUAD   (used 8/9 entries) of OT_QUAD
 Id 0  Operation      Left          Right          Third          Result
---- ---------- ------------ ------------ ------------ ------------
 3/ 0          ! TBL_VAR   3/ 0  < none >       < none >       TBL_TYPE   0/ 1
 3/ 1     return TBL_TYPE  0/10 TBL_CONST  3/ 0  < none >       TBL_TYPE   0/10
 3/ 2         if TBL_QUAD  3/ 0 TBL_QUAD   3/ 1 TBL_QUAD   3/ 7  < none >
 3/ 3          - TBL_VAR   3/ 0 TBL_CONST  3/ 0  < none >       TBL_TYPE   0/10
 3/ 4       push TBL_TYPE  0/10 TBL_QUAD   3/ 3  < none >       TBL_TYPE   0/10
 3/ 5       call TBL_VAR   1/ 0  < none >       < none >       TBL_TYPE   0/10
 3/ 6          * TBL_VAR   3/ 0 TBL_CODE   3/ 1  < none >       TBL_TYPE   0/10
 3/ 7     return TBL_TYPE  0/10 TBL_QUAD   3/ 6  < none >       TBL_TYPE   0/10


Block 0 4  main              (Level:  2)
  bl_type:   ST_FUNC
  bl_return:  default
  bl_parent: external
  bl_formal: <none>

TBL_CONST  (used 2/9 entries) of OT_TYPE
Note: '!' after Size - full access requires far pointer.
 Id0  Name             Class       Size       References
---- ---------------- ---------- ---- ----------------
 4/ 0  long constant   TY_CONST      4       Value: 10
 4/ 1  string constant TY_CONST     28       Value: "the factorial of 10 is %ld\n"

TBL_CODE   (used 3/9 entries) of OT_TYPE
 Id0  Quadruple Sequence
---- ------------------
 4/ 0     001100000004 TBL_CODE  4/ 1

 4/ 1     002100000003 TBL_QUAD  4/ 2 TBL_QUAD  4/ 3 TBL_QUAD  4/ 4

 4/ 2     003100000002 TBL_QUAD  4/ 0 TBL_QUAD  4/ 1


TBL_QUAD   (used 5/9 entries) of OT_QUAD
 Id 0  Operation      Left          Right          Third          Result
---- ---------- ------------ ------------ ------------ ------------
 4/ 0       push TBL_TYPE  0/10 TBL_CONST  4/ 0  < none >       TBL_TYPE   0/10
 4/ 1       call TBL_VAR   1/ 0  < none >       < none >       TBL_TYPE   0/10
 4/ 2       push TBL_TYPE  1/ 2 TBL_CONST  4/ 1  < none >       TBL_TYPE   1/ 2
 4/ 3       push TBL_TYPE  0/10 TBL_CODE   4/ 2  < none >       TBL_TYPE   0/10
 4/ 4       call TBL_VAR   1/ 2  < none >       < none >       TBL_TYPE   0/ 1
```

```
subroutine
 factorial,
 return_type   'signed long',
 called_by
   '',
 calls
   factorial,
 params
   'signed long i',
 ext_vars
   '',
 loc_vars
   '',
 metrics
   '',
{
 if (!i)
   return (1);
 else
   return (i * factorial(i-1);
}
subroutine
 main,
 return_type default,
 called_by
   '',
 calls
   factorial,
   printf,
 params
   '',
 ext_vars
   '',
 loc_vars
   '',
 metrics
   '',
{
 printf("the factorial of 10 is %ld\n", factorial (10));
}
```

The NSL Representation
Figure 4.4.4



The Resulting NS Diagrams
Figure 4.4.5

4.5 UNEXPECTED RESULTING TOOLS

There were two results of this research that were not expected. A "Railroad" Diagrammer was developed, and the NS Diagrammer was useful as a stand-alone processor.

4.5.1 RAILROAD DIAGRAMMER

Debugging a grammar written in Backus-Naur Form (BNF) can be a tricky process. The grammar for the DL, although based on X3J11 C, was written "from scratch". In some cases, it appeared that the productions were not being processed as expected. It was determined that processing and printing the grammar in an 'order used' manner might prove useful in finding the problem. This utility was written, and when the output was inspected, it appeared that automatic generation of railroad, or syntax, diagrams would be a straightforward "reformatting" of the output.

4.5.1.1 Railroad Diagrammer Strategy

In the initial output, five classes of objects were generated that appear in Railroad Diagrams. These were tokens, punctuation, productions, empty alternatives, and recursive productions where:

- Tokens are the string of letters, and digits that represent keywords and identifiers;
- Punctuation refers to typically single character tokens that do not fit the rules that recognize regular tokens;
- Productions are rule based sequences of tokens;
- Empty alternatives are used within a production when a token, or production, is optional and may be omitted; and
- Recursive productions are productions that allow for multiple successive occurrences of themselves.

Two more objects: continuation lines and off page connectors, were then added to nicely display the grammar rules that had alternatives that did not fit on one line, or rules too big to fit on one page.

4.5.1.2 Railroad Diagrammer Implementation

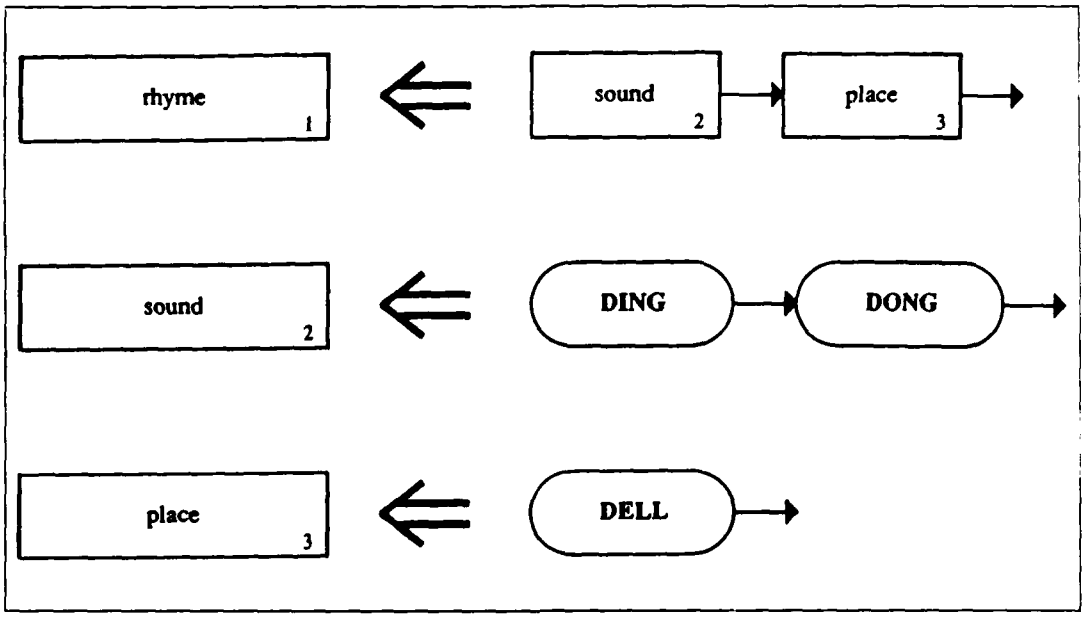
First, each object was described in the graphic display language Postscript from Adobe Systems. Then these Postscript descriptions were collected into a library. The Diagrammer was built as a two-phase system, where the first phase consumed the rules one at a time, parsed them into the proper collection of objects, and constructed another file containing calls to the library routines. Once the complete Postscript description of the grammar had been generated, it was processed by the Postscript engine on a laser printer, yielding the pictorial representation of the grammar.

4.5.1.3 Railroad Diagrammer Example

A short example of the Railroad Diagrammer operation is contained in Figures 4.5.1.3.1 and 4.5.1.3.2 below, and is taken from the documentation of yacc. [Schrei] For a more complete example of a Railroad Diagram, the reader is referred to Appendix A of this report, which contains the Railroad Diagrams for the DL.

```
%token DING DONG DELL
%%
rhyme
    : sound place
    ;
sound
    : DING DONG
    ;
place
    : DELL
    ;
```

Example Grammar Specification
Figure 4.5.1.3.1



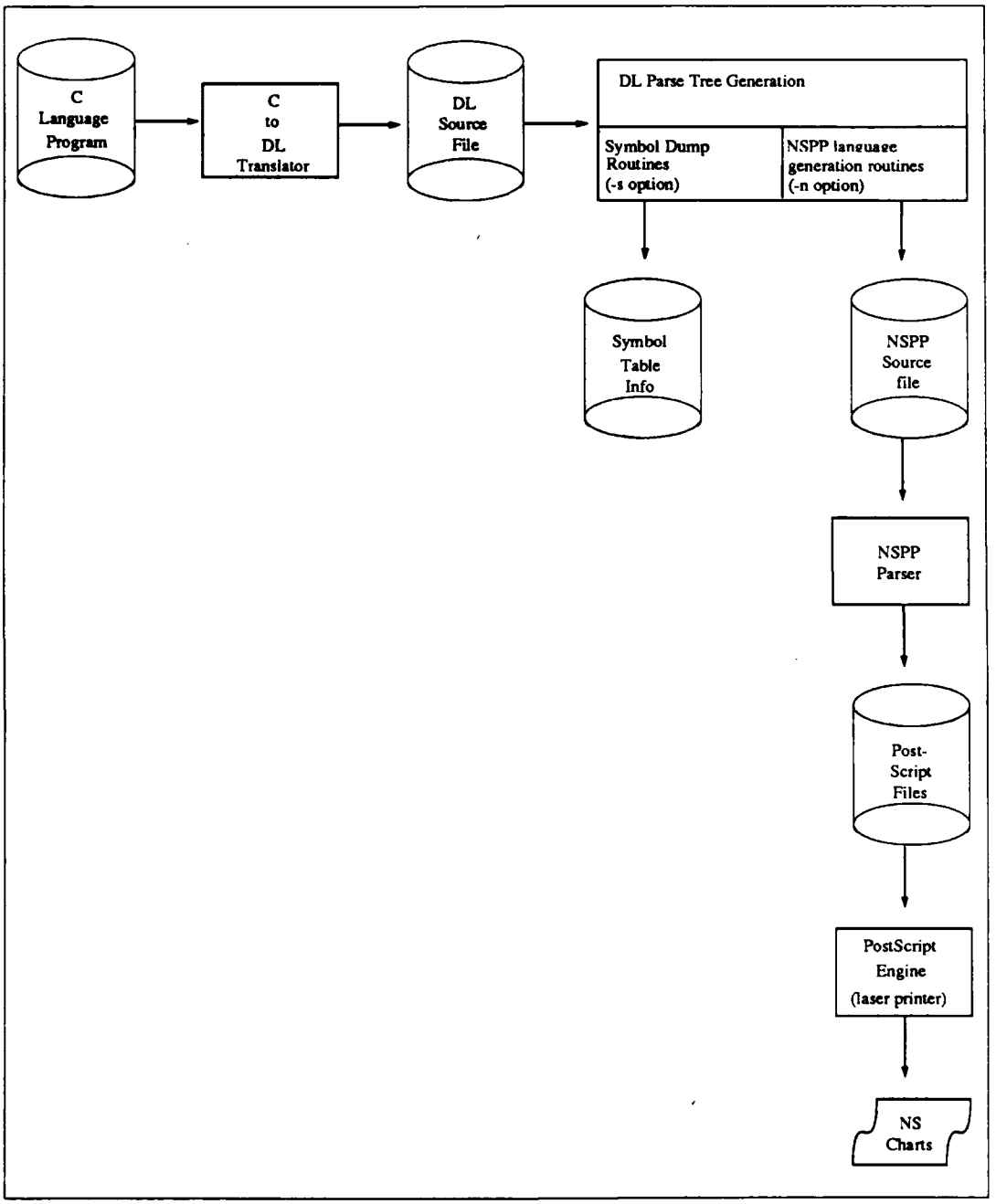
Railroad Diagram For Example Grammar
Figure 4.5.1.3.2

4.5.2 STAND-ALONE NS DIAGRAM GENERATOR

Because of the way the NSL processor was implemented (see Section 4.3, Postprocessor, for a detailed description), it operates independently. This has proven to be quite useful as an alternative way to generate user-level logic diagrams. Unfortunately, most users still feel more comfortable with the older unstructured Flowcharts.

4.6 SUMMARY OF IMPLEMENTATION

As Figure 4.6.1 below portrays the research prototype is a multiphase system, built to allow for the examination of intermediate results as they are generated. Every file built (depicted with the barrel symbol) is capable of producing output to a monitor, or to hardcopy. This feature allowed for the progress to proceed in an organized, well-tested fashion, knowing each step was built on correct input from the previous one.



Research Prototype Dataflow Diagram
Figure 4.6.1

CHAPTER 5 - CONCLUSIONS

5.1 FINDINGS

To reiterate, the goals of this research project were to:

1. Research documentation techniques specifically appropriate to the software maintenance environment, as contrasted with methods commonly used in development;

2. Summarize the programming language features that were appropriate for the class of potential source languages for the documentation tool;

3. Research the design of a general documentation language, focusing on the ability to handle a selected class of languages, using the proposed approach (structured in the three categories of preprocessor, compiler, and postprocessor); and

4. Determine if the proposed approach could be implemented to automatically transform programming language source code into documentation.

At this time, the end of the contract, we believe that we have met our goals as follows:

1. The research into the documentation methodologies which are currently in use successfully led to the knowledge of the documentation and output requirements perceived to be relevant to, and by, maintenance programmers;

2. The research into programming languages and language features has led to an understanding of the input requirements for a maintenance tool;

3. A general documentation language was developed, containing all of the features of the selected class of target languages. This language was implemented, i.e., a compiler was built for it, as the centerpiece of a prototype maintenance tool; and

4. The three phase design was partially implemented, the DL compiler is now essentially complete, and the postprocessor (for modified NS Diagrams) is also complete. A C to the DL preprocessor is complete, and the research and design for a FORTRAN to the DL preprocessor is also essentially complete. As such, the prototype accepts working C code and generates NS Diagrams for it.

The work on this project led to some interesting conclusions. The first of these was the determination that none of the methodologies in use today are really optimal in their utility to maintenance programmers. While NS Diagrams, and others, are familiar to programmers and to many knowledgeable users, the optimal tool would present a new methodology providing graphics, text, detail, overview, static structure, and dynamic dataflow together in a visually-appealing, user-friendly way.

The second conclusion is that the generality requirement is indeed a requirement, and not just a consideration of interest. Programmers working with different languages have widely different perceptions about what is useful documentation.

5.2 BENEFITS FROM THE WORK

Several unexpected tools were revealed during the implementation phase. These tools have proven useful, and have actually been used to locate errors. Postprocessors have proven to be relatively easy to implement, due to the partitioned approach, which converts the compiled structures into documentation. Further, as designed, the first stage of the NS postprocessor generates information which is useful (as is) to programmers, or which could be used by another postprocessor that was designed to display similar information.

A grammar bug was discovered in TSI's Development Environment for Efficient Programming (DEEP) by examining the output of the Railroad Diagrammer. TSI's need for generating program logic diagrams may have been automated, providing a switch from standard Flowcharts to Nassi-Sh neidermann (NS) Diagrams.

Additionally, it was interesting to learn that out of the plethora of documentation techniques available, most were not particularly suitable in the maintenance environment. Most documentation techniques are oriented toward the design of new software applications. Top-down design techniques treat the design phase similar to an outline of the application to be implemented, where the details are filled in after the architecture of the applica ion is completed.

As such, desigr documents frequently gloss over details to provide as abstract a view as possible, whereas the resulting programs are more concerned with implementing something that behaves according to the abstract design. Generally, once low level detail is added to a design, it becomes difficult (if not impossible) to separate out those statements representing abstract high level structures from low level implementation details.

In a third generation language, such as those we considered, the segregation of high level abstract information from low level details would require some form of marking, or signalling the different components (high level versus low level elements). Without introducing some form of signalling into the original program (the one being documented), this separation of abstract code from detail code, is impossible.

Without separation of abstract and low level detail, only detailed or low level documentation results. This is useful in its own right, but does little to introduce an unfamiliar maintenance programmer to the code being documented, particularly since the detail is still overwhelming. Low level documentation has the most value, because when examining a program in a different form aids in finding the errors introduced by coding style, or misunderstood language constructions.

For example, during this project, programming errors were revealed because the documenting process ignored "visual" layout clues which mislead proper interpretations of the true meaning of the code. Cases in which this type of error occurred included evaluation of nested if/then/else statements and grammar rules (in the grammar of a compiler).

5.3 PROBLEMS WITH THE APPROACH

At this time, we perceive two problems with the research. First, the prototype operation is batch-oriented, not interactive, and consequently is not perceived by programmers as optimally usable in a real world environment.

Second, no "validation" of the approach and prototype has been performed. This validation must include allowing real world programmers to use the prototype output, and determining how well that output facilitated their understanding of the code. This would require a significant amount of additional research involving the development of a user-interface.

5.4 FUTURE STUDY AREAS

There are two obvious areas for future study remaining at this time. The first area is the use of graphics in the maintenance environment. When this project began, the Macintosh was not yet a viable system and windows were unavailable on other systems. The use of graphics and windows should be examined with regard to hypertext-oriented applications that would provide on-line, interactive documentation.

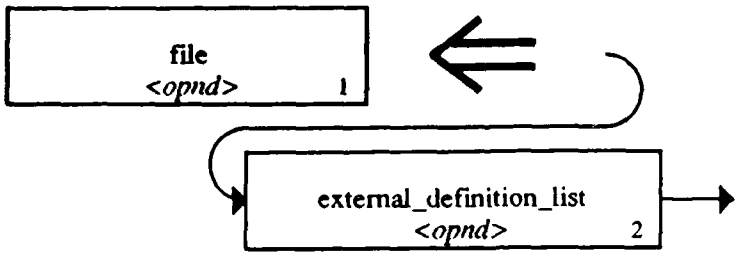
The second area involves further development of the concept of relationships between grammars. Given that a finer definition of concepts common to all languages can be made, it should be possible to develop a "table-driven" translator. In such a case, a tabular mapping of the common concepts into the DL would serve as the foundation for all individual source language to the DL translators. These individual translators, whether FORTRAN, Pascal, etc., would access the table for the treatment of the canonical concepts, and would use ad hoc treatments for concepts unique to that language.

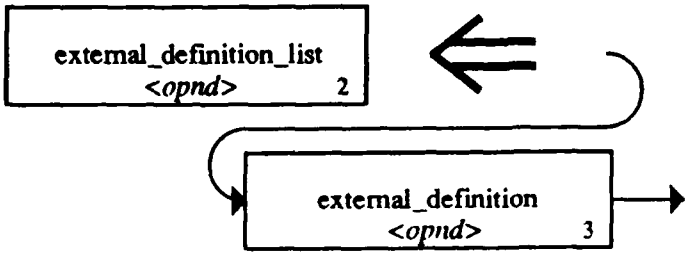
These translators would be designed to generate "good", idiomatic DL, not the brute force rendering common to the translators available today. Obviously, if "good" DL could be generated, then it should be possible to replace the DL code with code specific to another programming language, that is, generating good, idiomatic C or Pascal from FORTRAN.

APPENDIX A

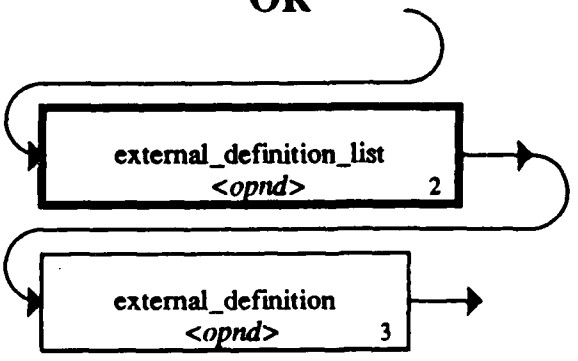
DL RAILROAD DIAGRAMS

RAILROAD DIAGRAMS for DL GRAMMAR



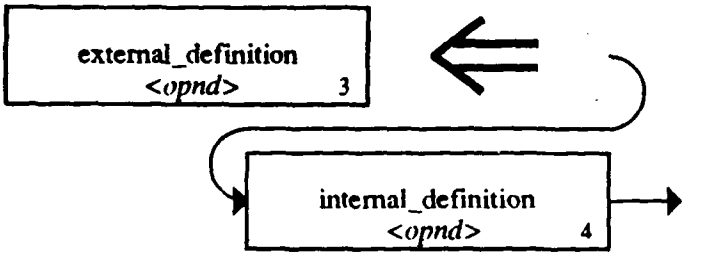


OR

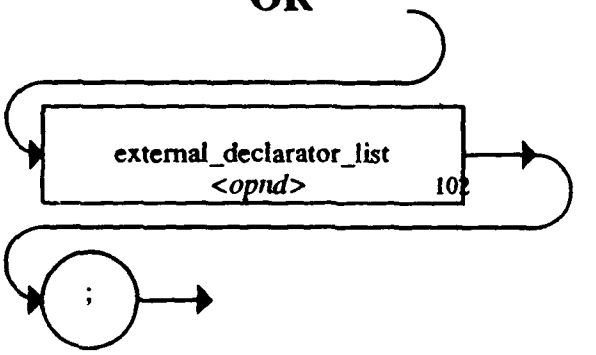


NOTE: Heavy bordered box denotes recursive definition.

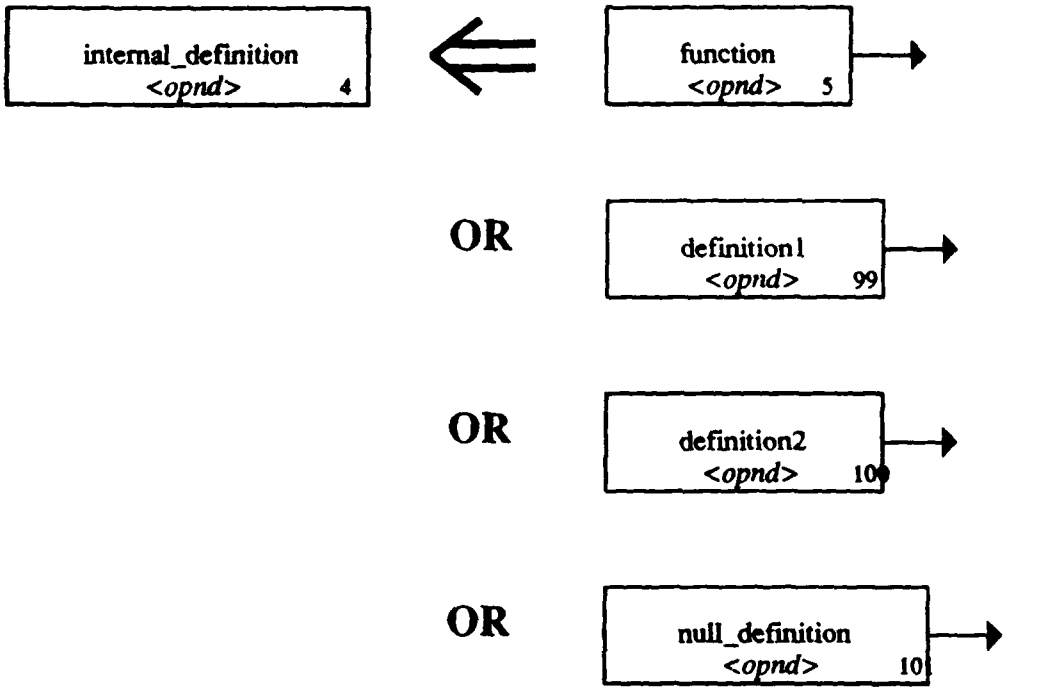
RAILROAD DIAGRAMS for DL GRAMMAR

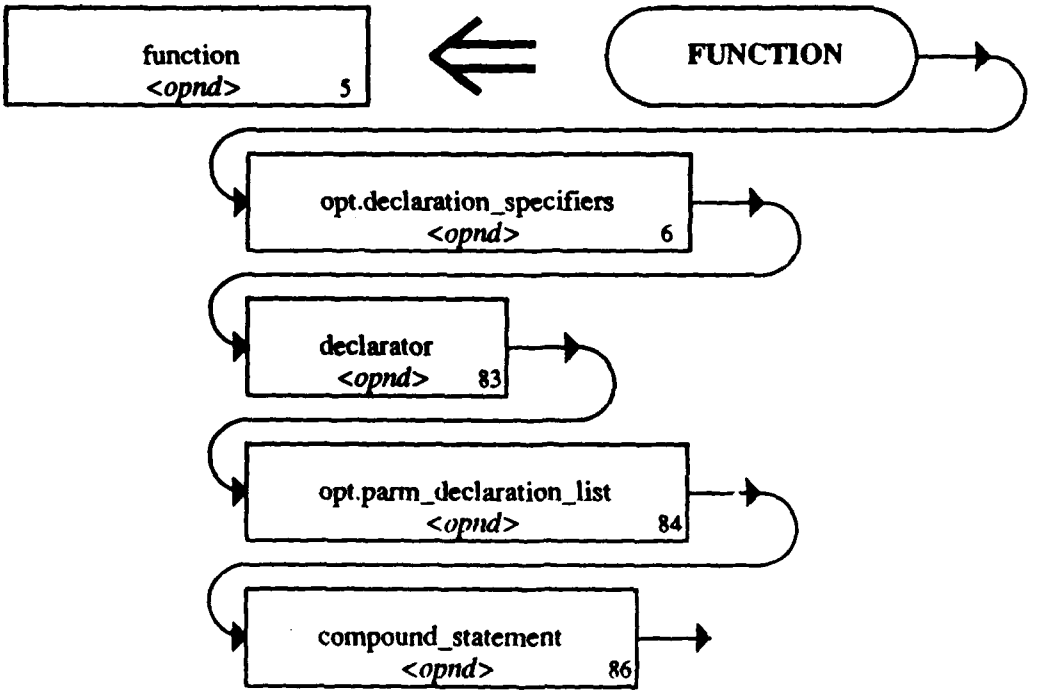


OR



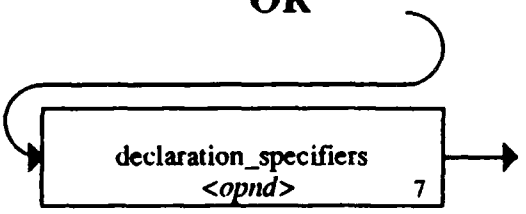
RAILROAD DIAGRAMS for DL GRAMMAR

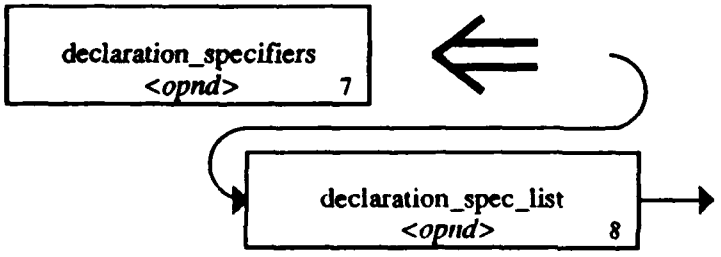




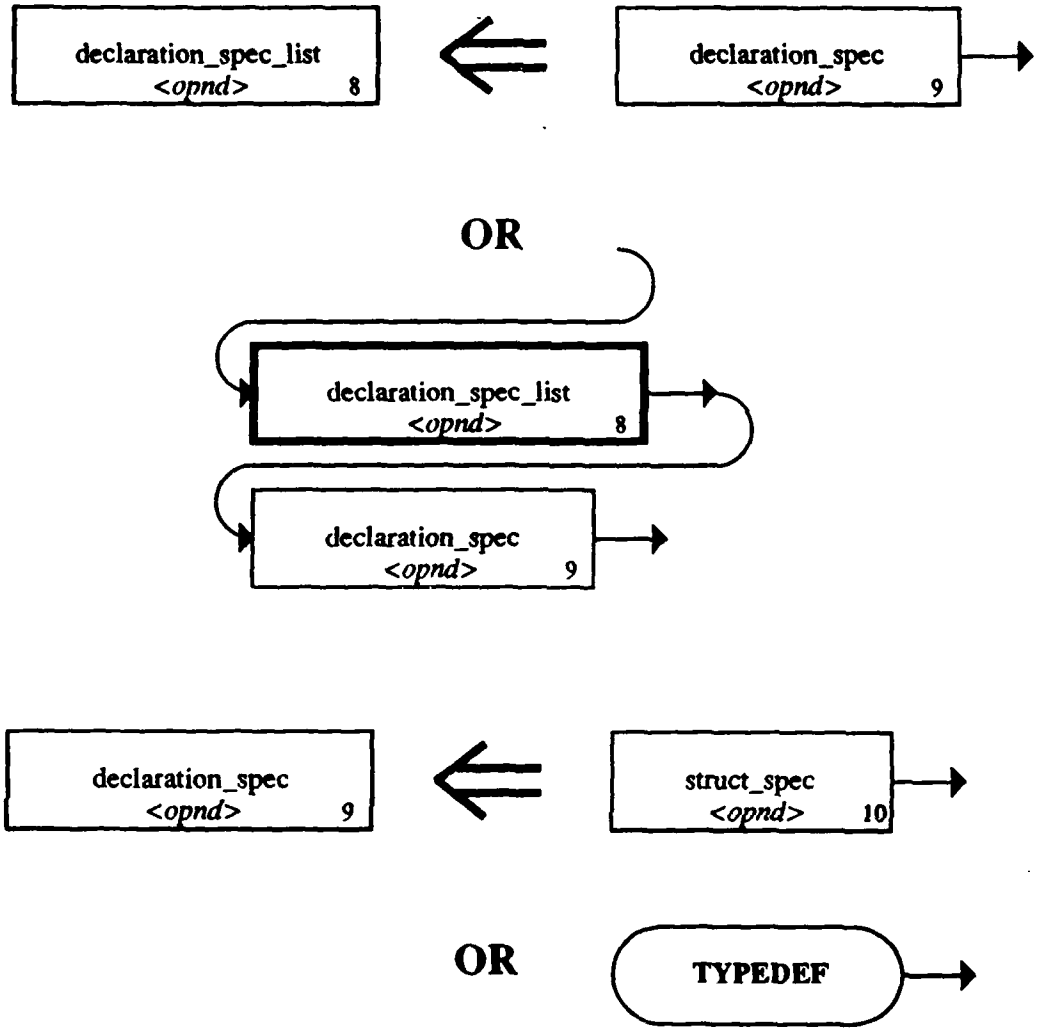
RAILROAD DIAGRAMS for DL GRAMMAR

OR

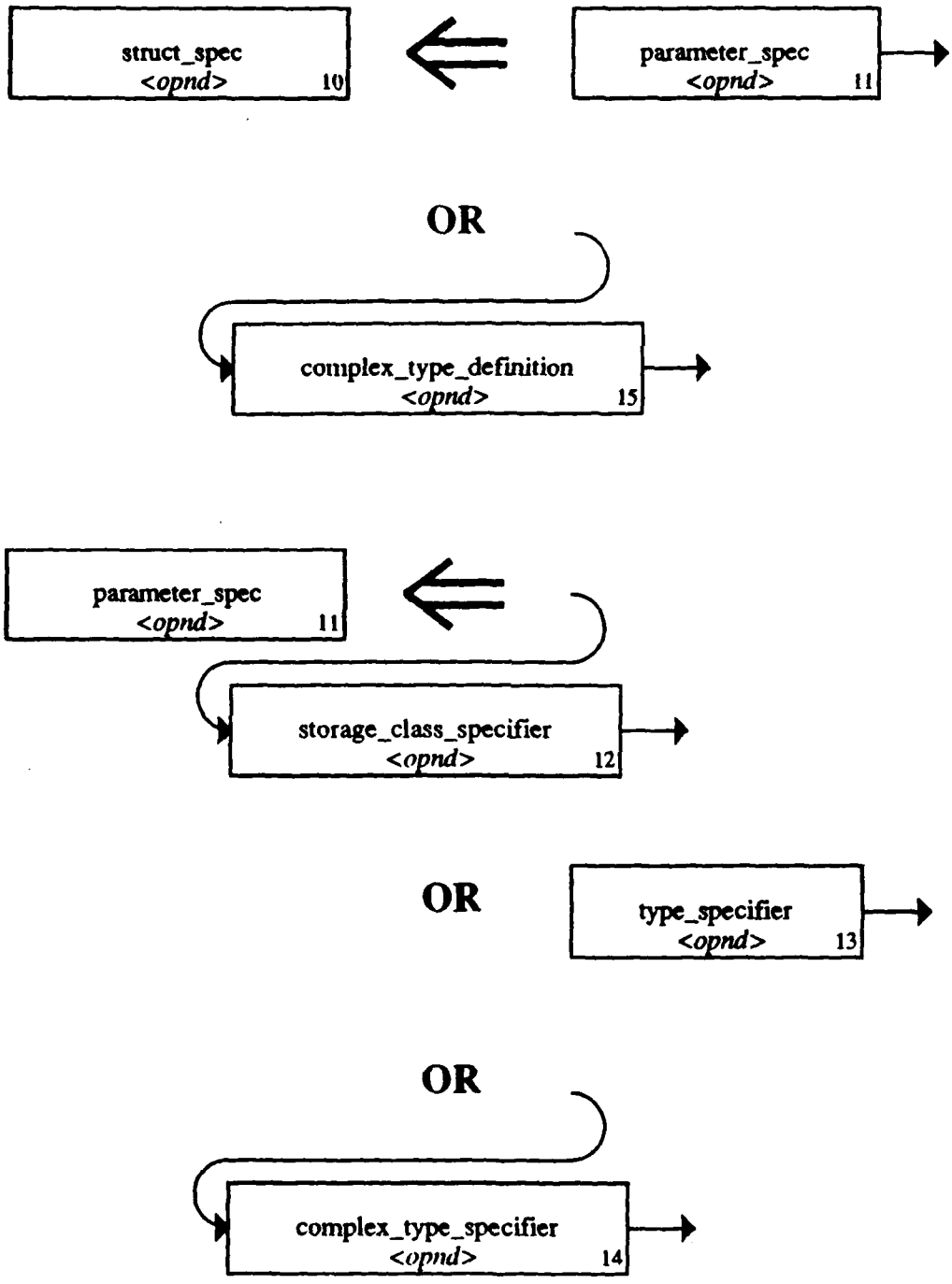




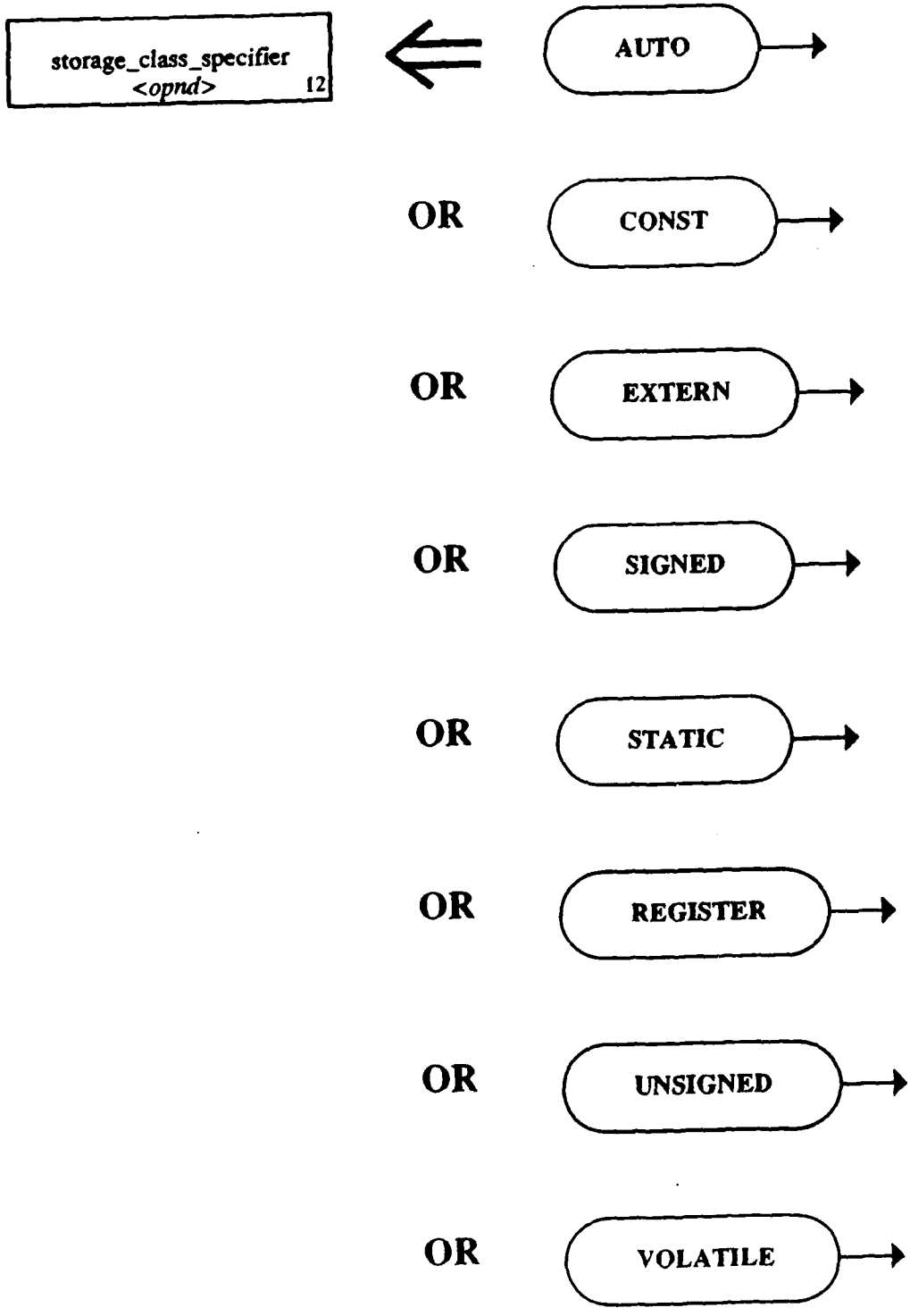
RAILROAD DIAGRAMS for DL GRAMMAR



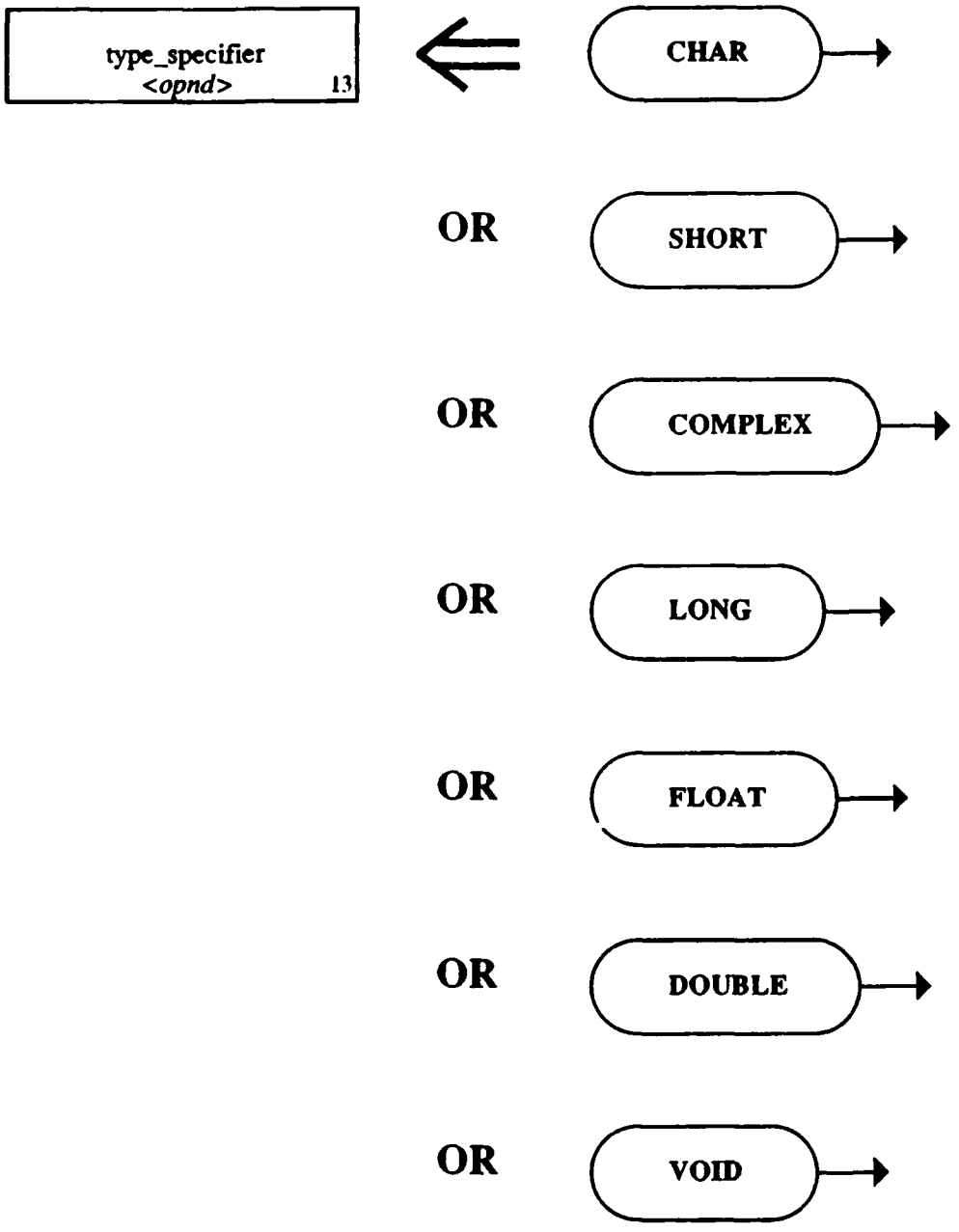
RAILROAD DIAGRAMS for DL GRAMMAR



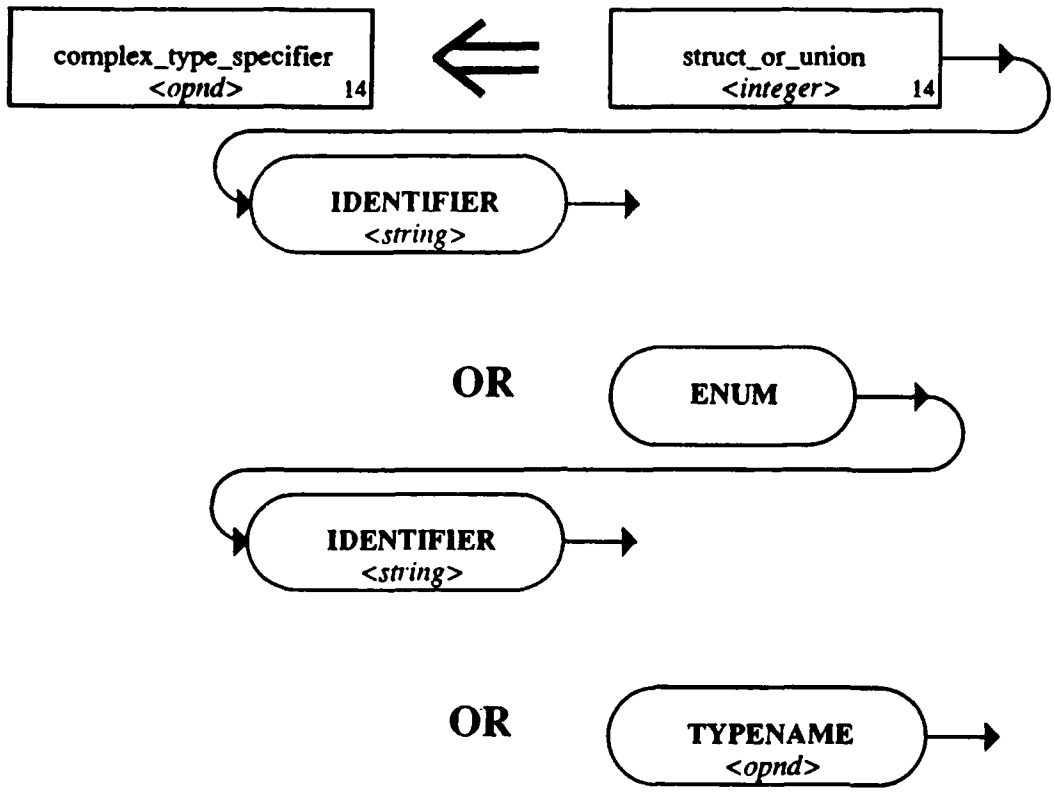
RAILROAD DIAGRAMS for DL GRAMMAR



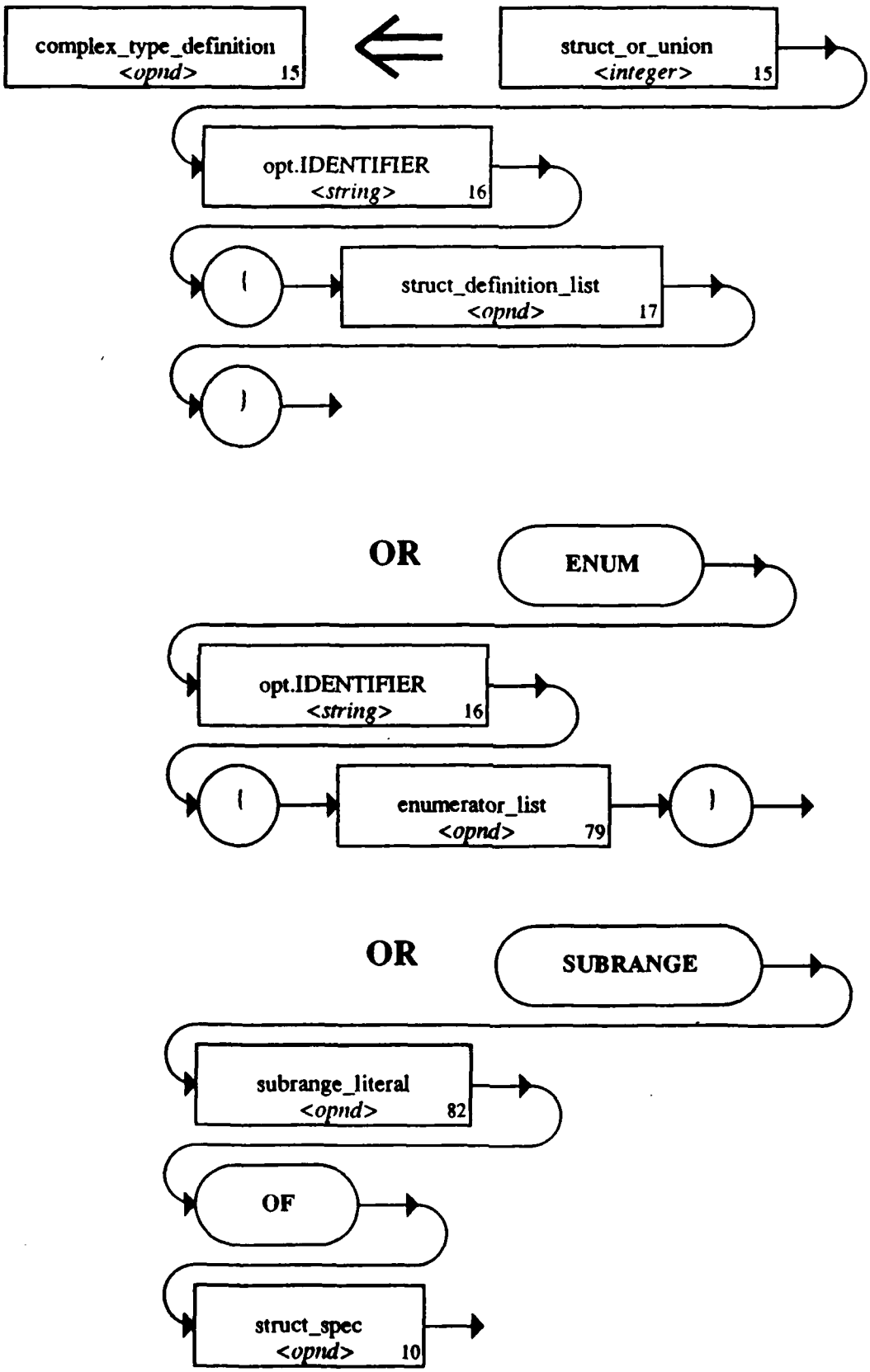
RAILROAD DIAGRAMS for DL GRAMMAR



RAILROAD DIAGRAMS for DL GRAMMAR



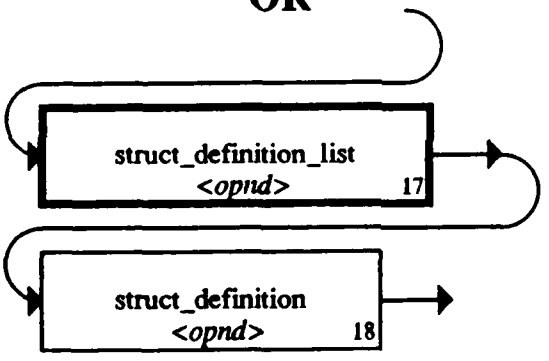
RAILROAD DIAGRAMS for DL GRAMMAR



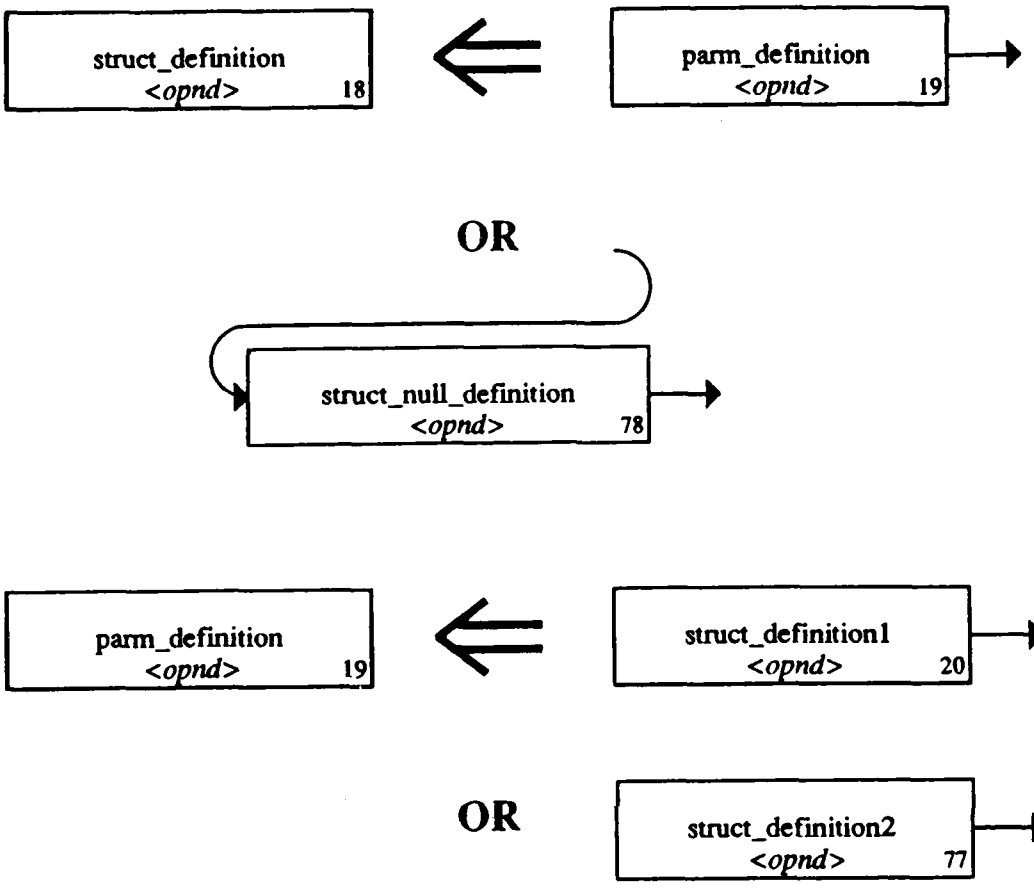
RAILROAD DIAGRAMS for DL GRAMMAR

OR

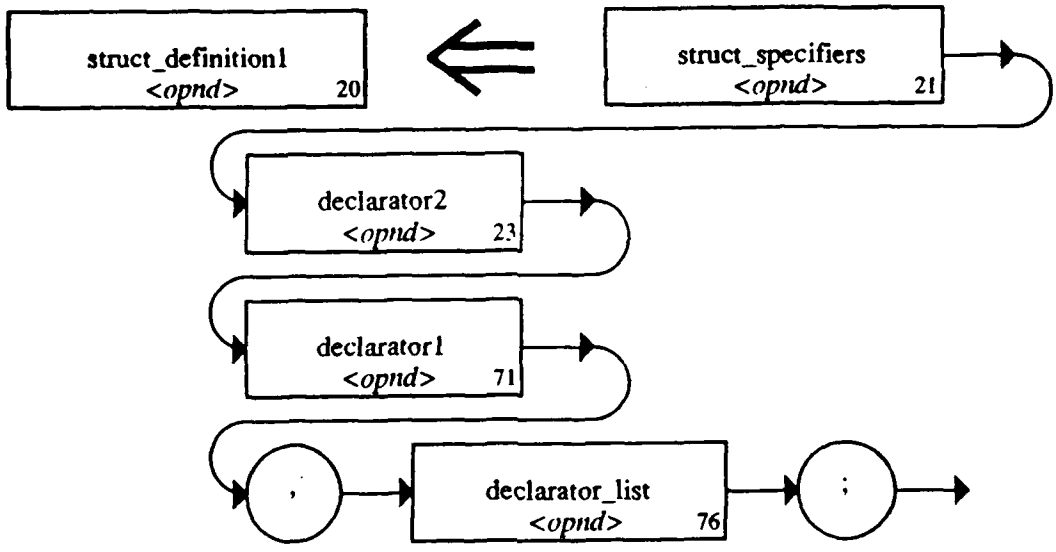
OR

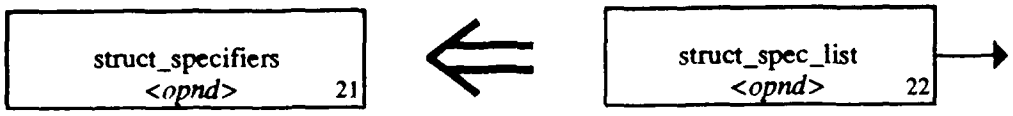


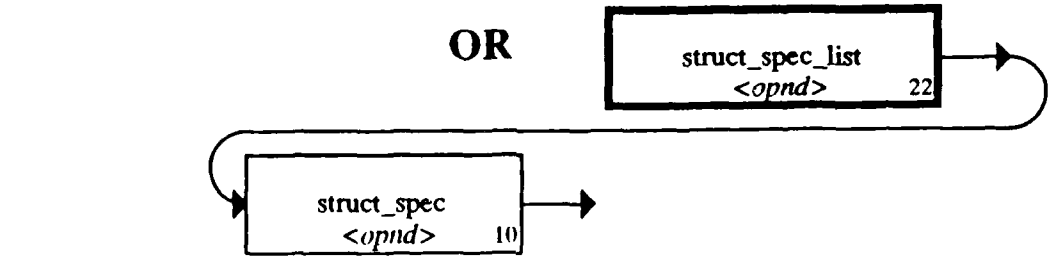
RAILROAD DIAGRAMS for DL GRAMMAR



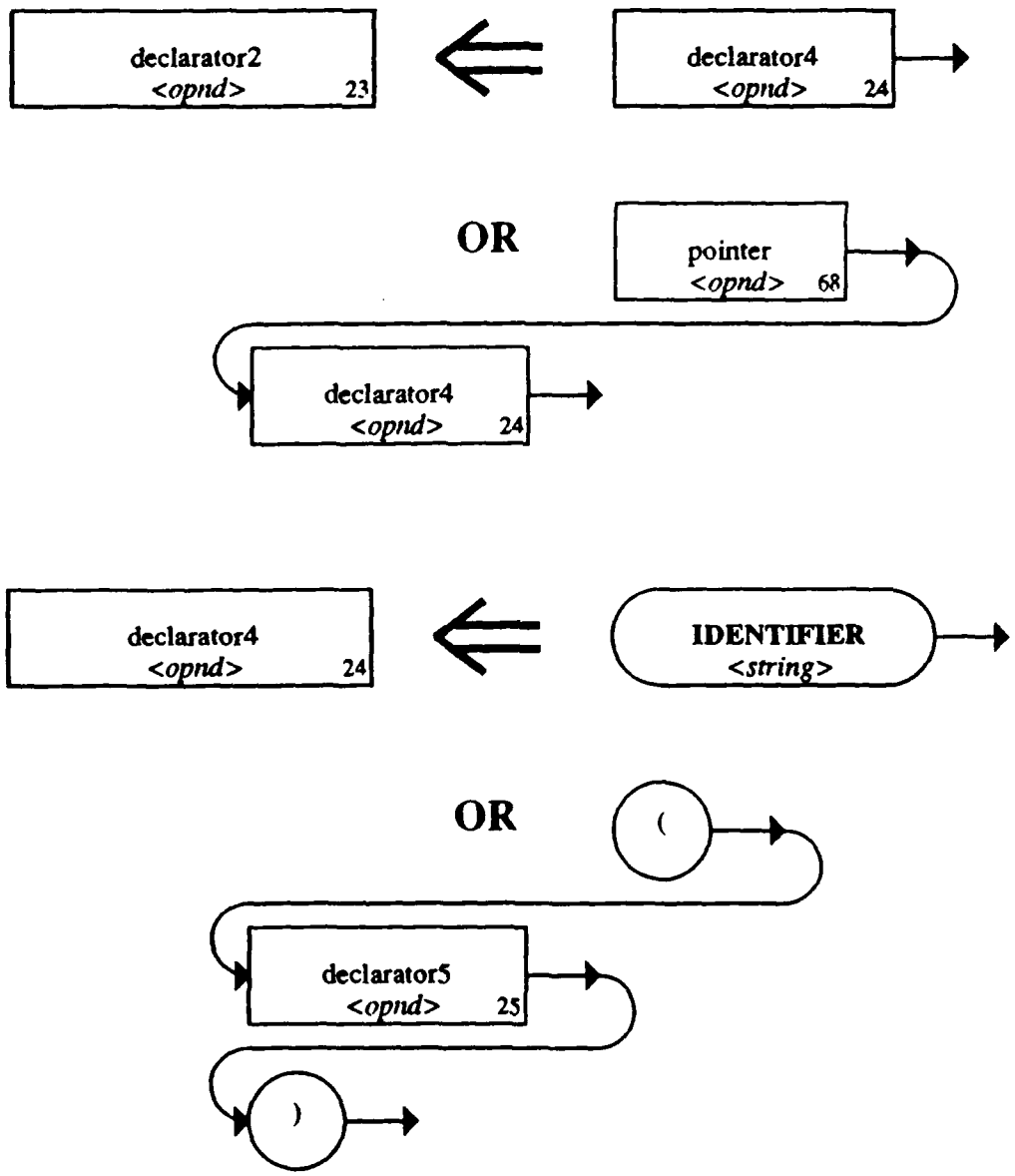
RAILROAD DIAGRAMS for DL GRAMMAR



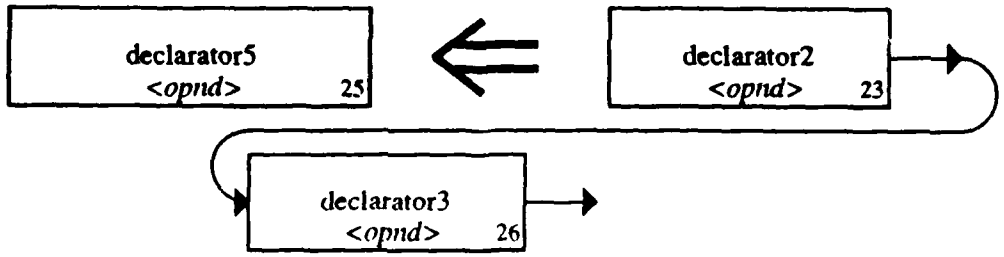




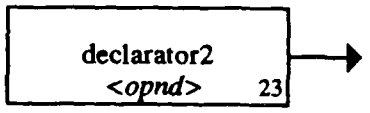
RAILROAD DIAGRAMS for DL GRAMMAR



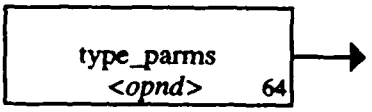
RAILROAD DIAGRAMS for DL GRAMMAR



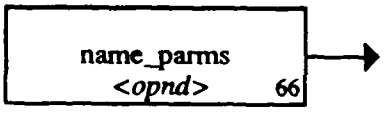
OR



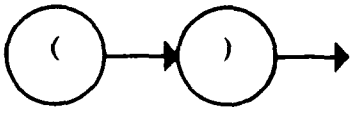
OR



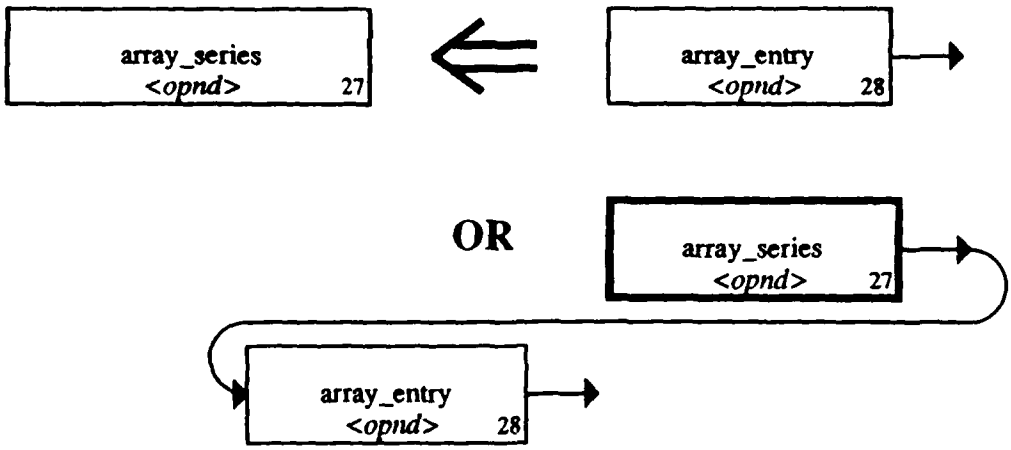
OR



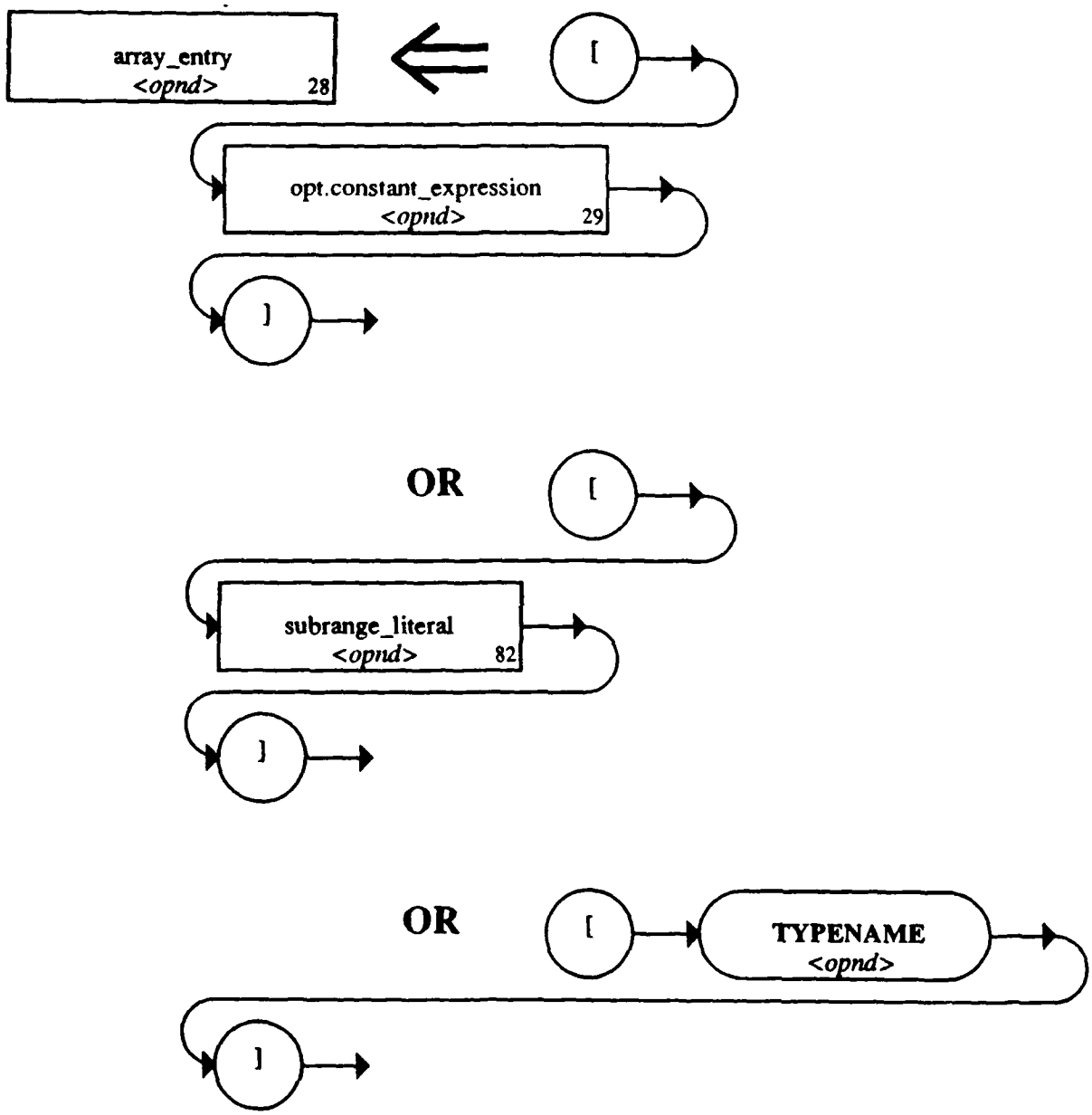
OR



RAILROAD DIAGRAMS for DL GRAMMAR

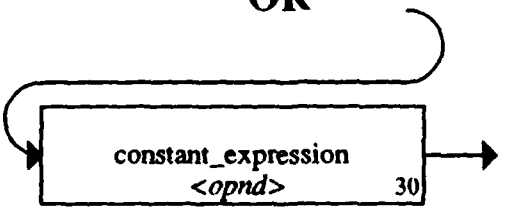


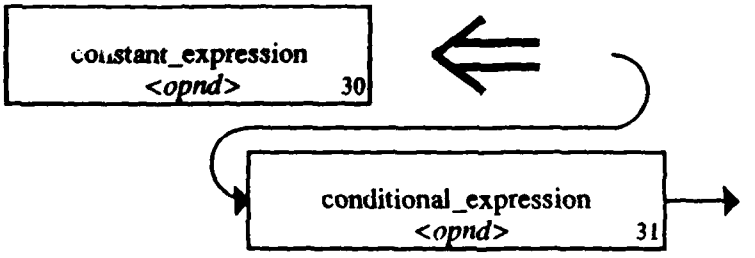
RAILROAD DIAGRAMS for DL GRAMMAR



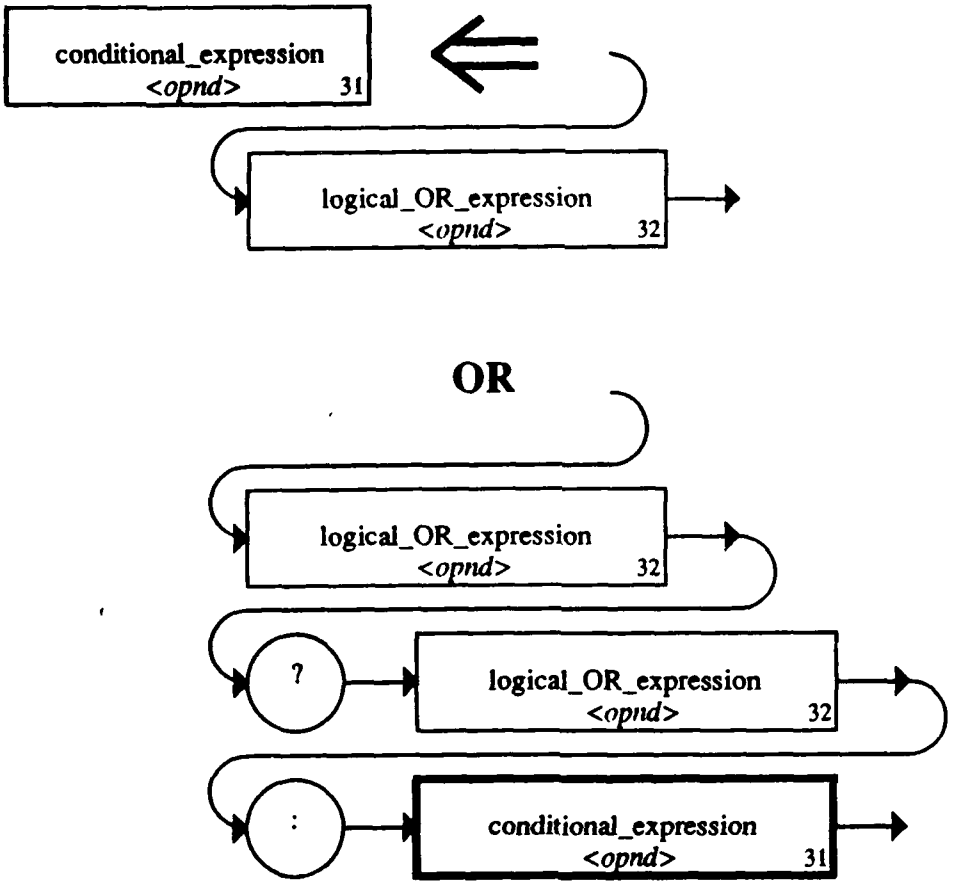
RAILROAD DIAGRAMS for DL GRAMMAR

OR

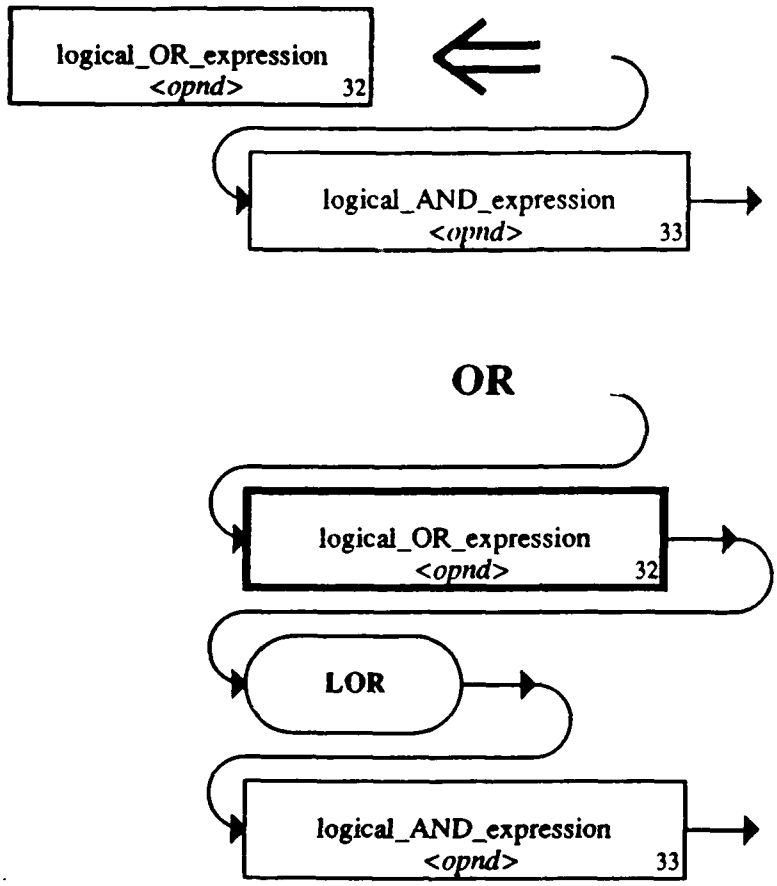




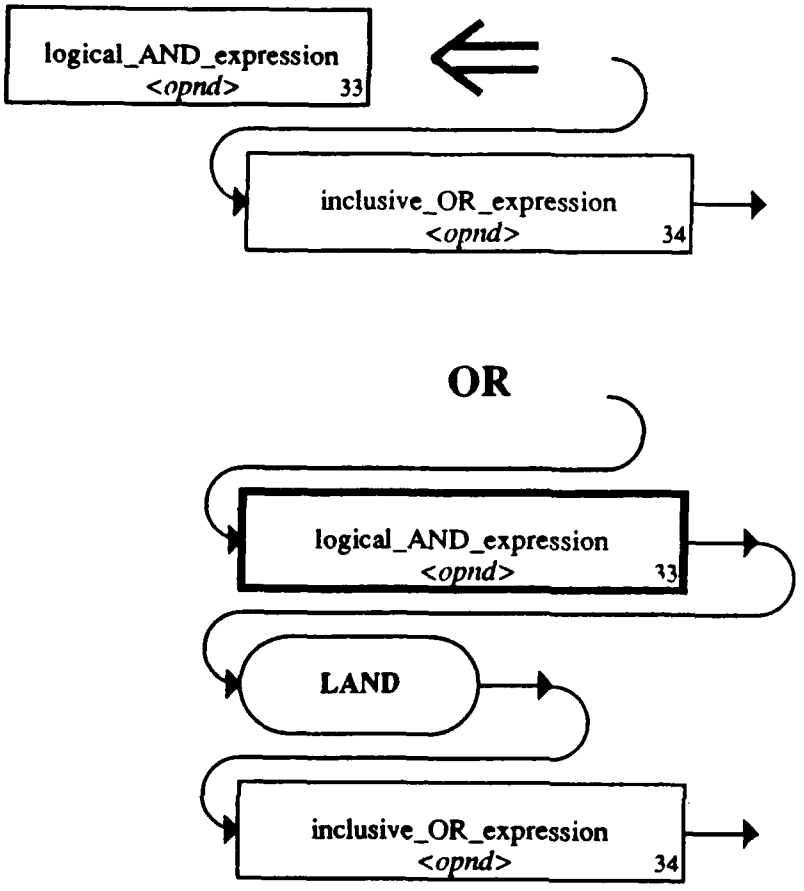
RAILROAD DIAGRAMS for DL GRAMMAR



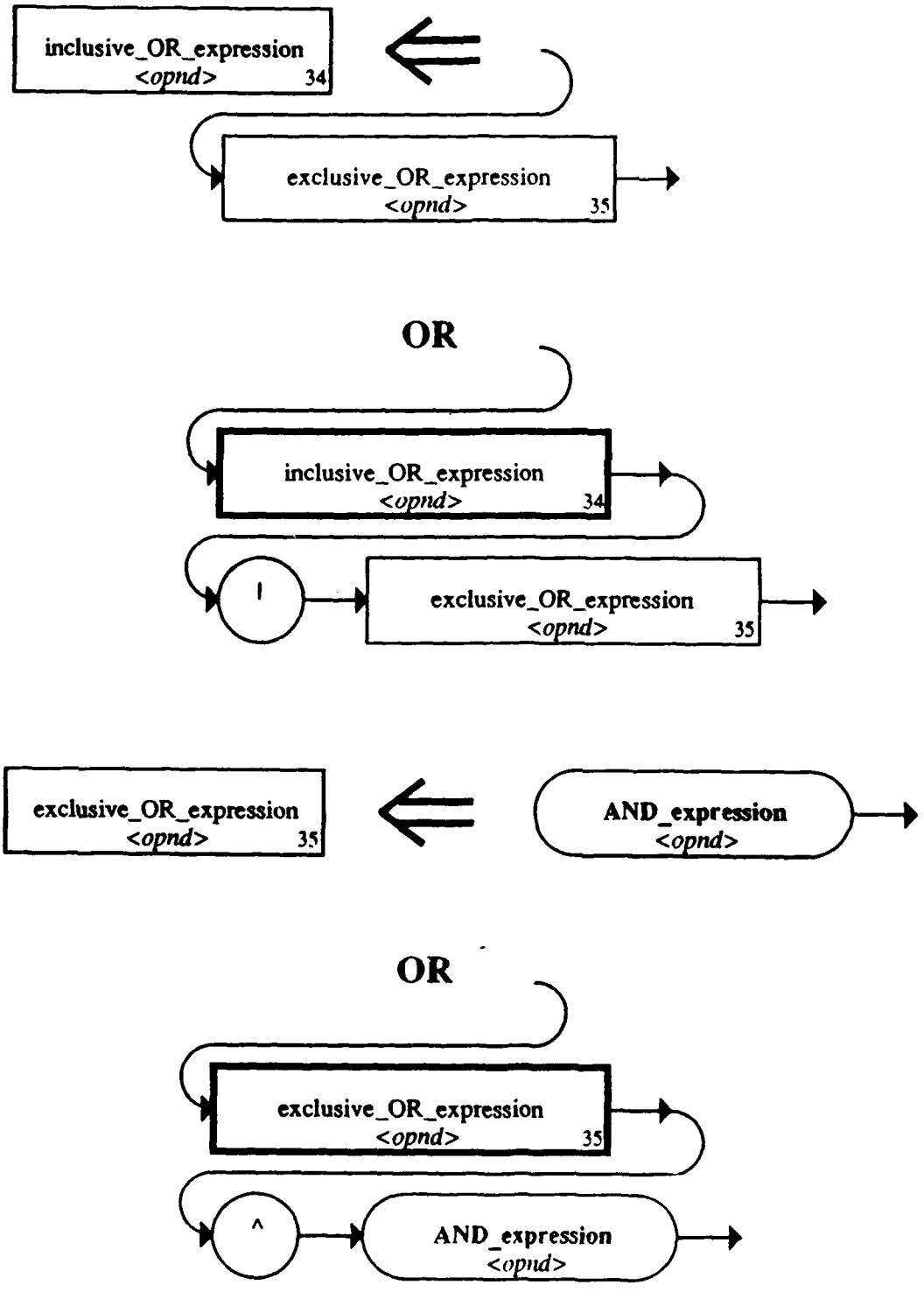
RAILROAD DIAGRAMS for DL GRAMMAR



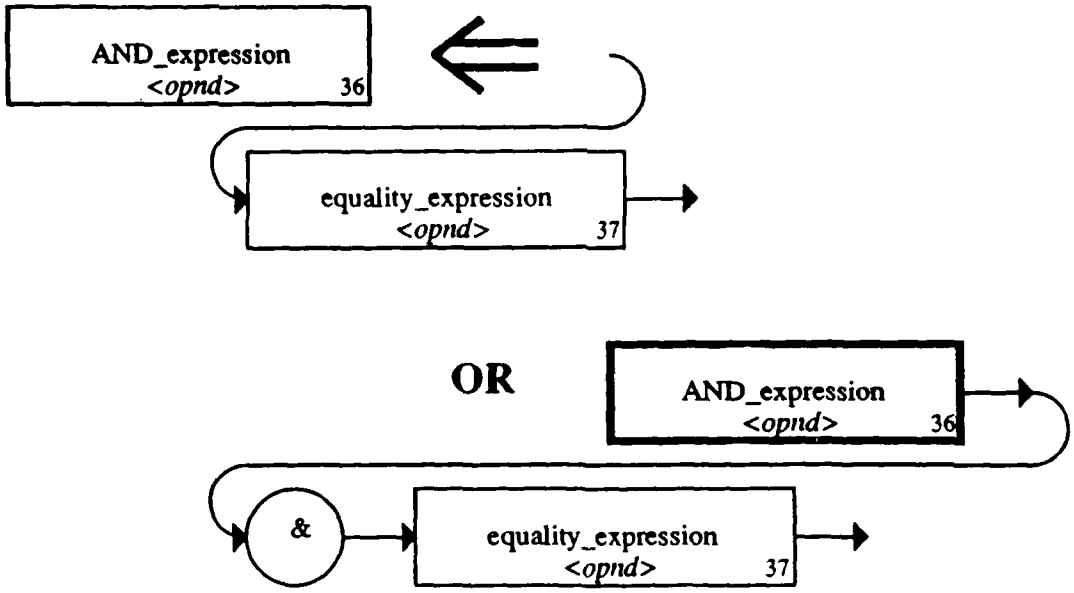
RAILROAD DIAGRAMS for DL GRAMMAR



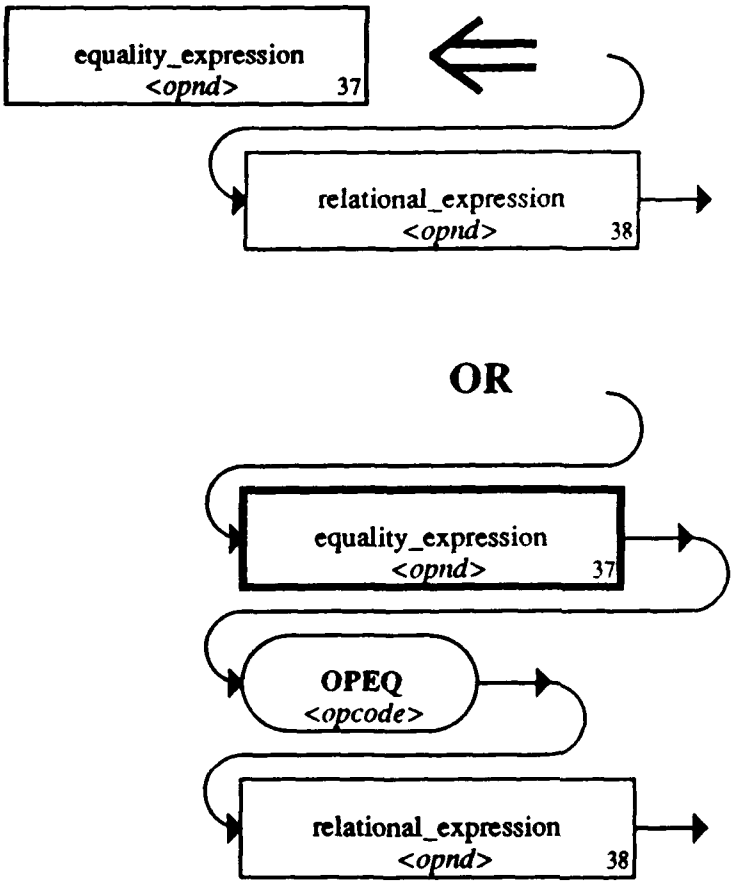
RAILROAD DIAGRAMS for DL GRAMMAR



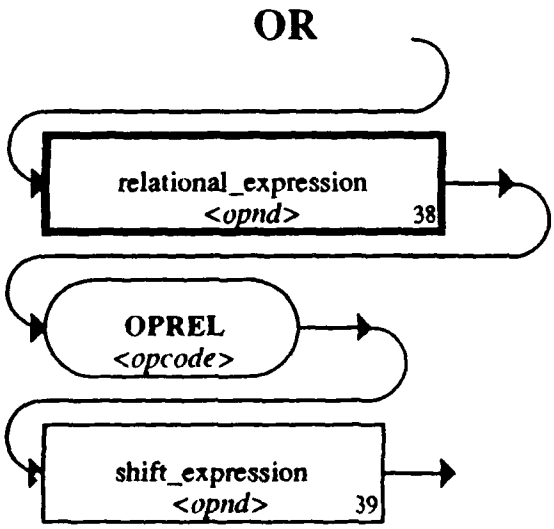
RAILROAD DIAGRAMS for DL GRAMMAR



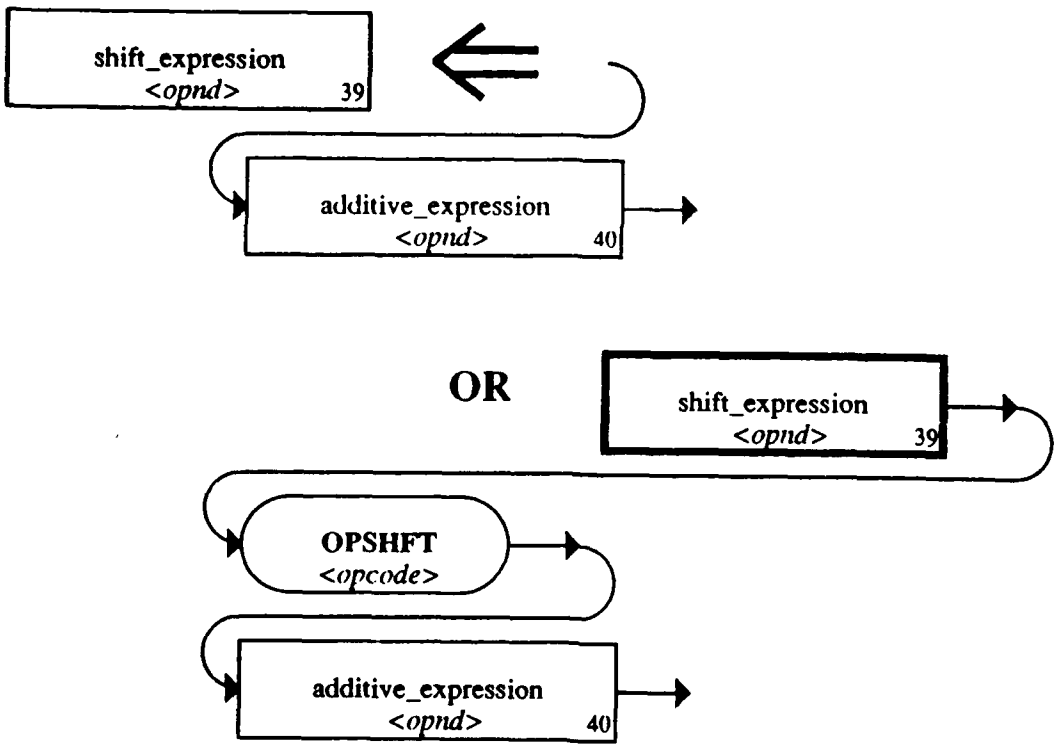
RAILROAD DIAGRAMS for DL GRAMMAR



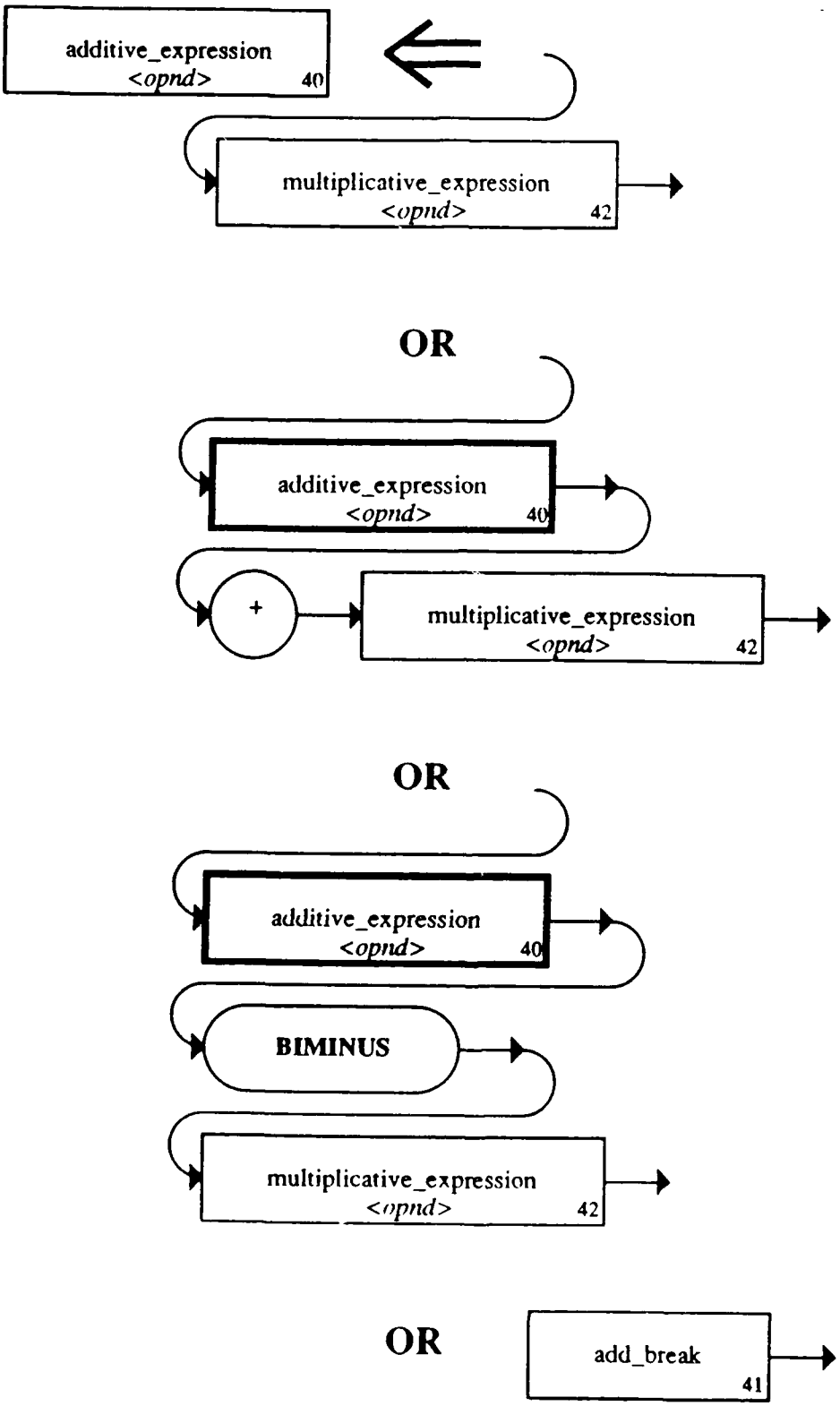
RAILROAD DIAGRAMS for DL GRAMMAR



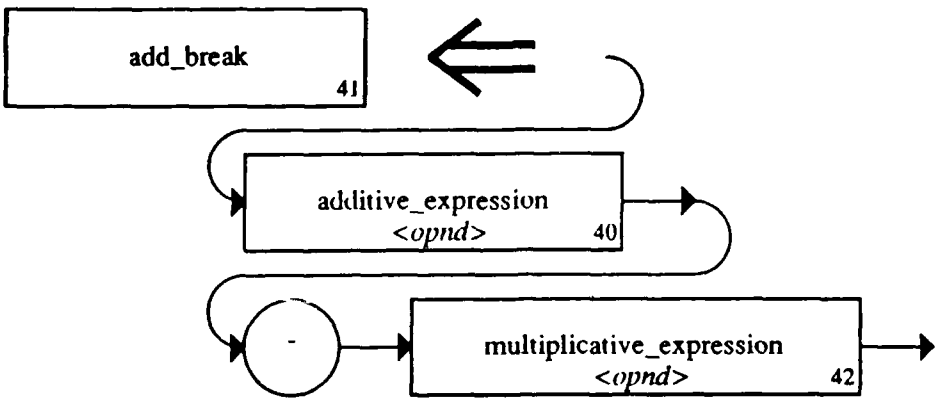
RAILROAD DIAGRAMS for DL GRAMMAR



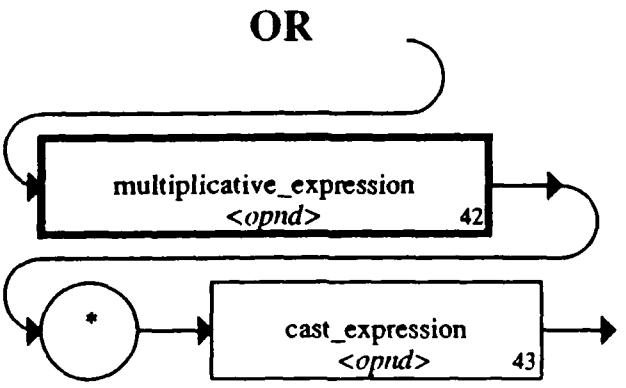
RAILROAD DIAGRAMS for DL GRAMMAR

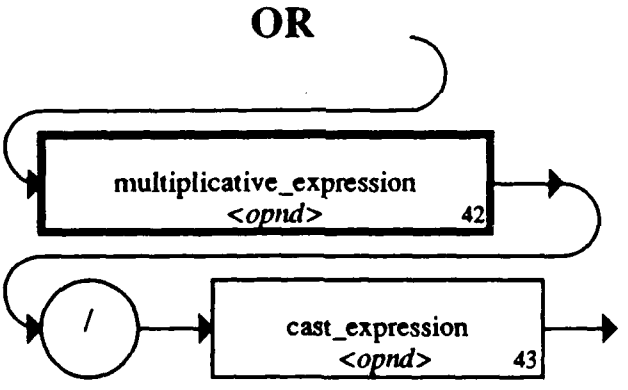


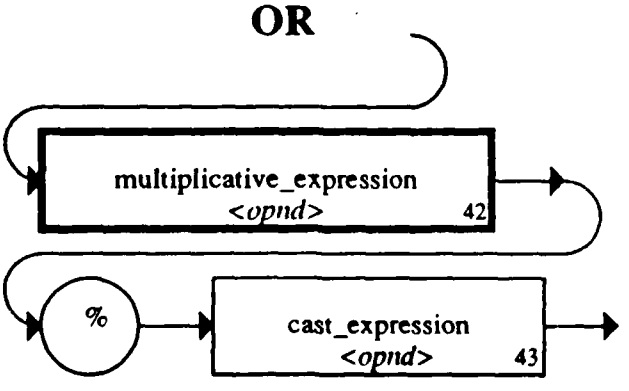
RAILROAD DIAGRAMS for DL GRAMMAR



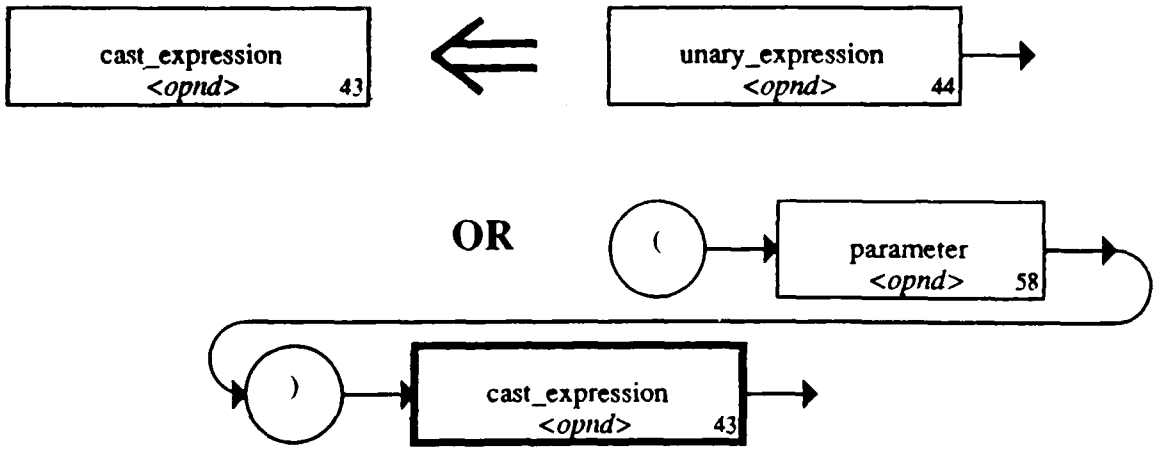
RAILROAD DIAGRAMS for DL GRAMMAR



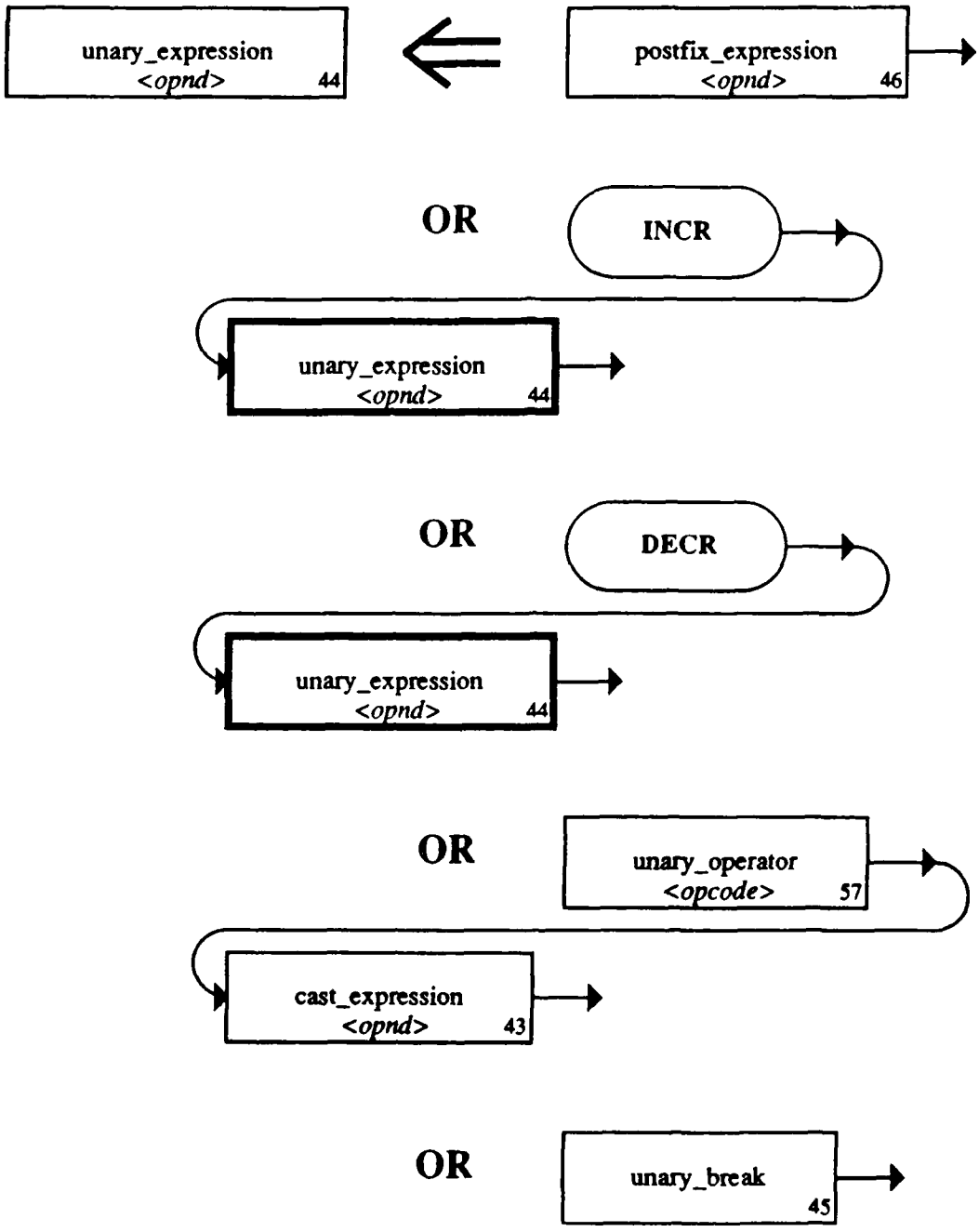




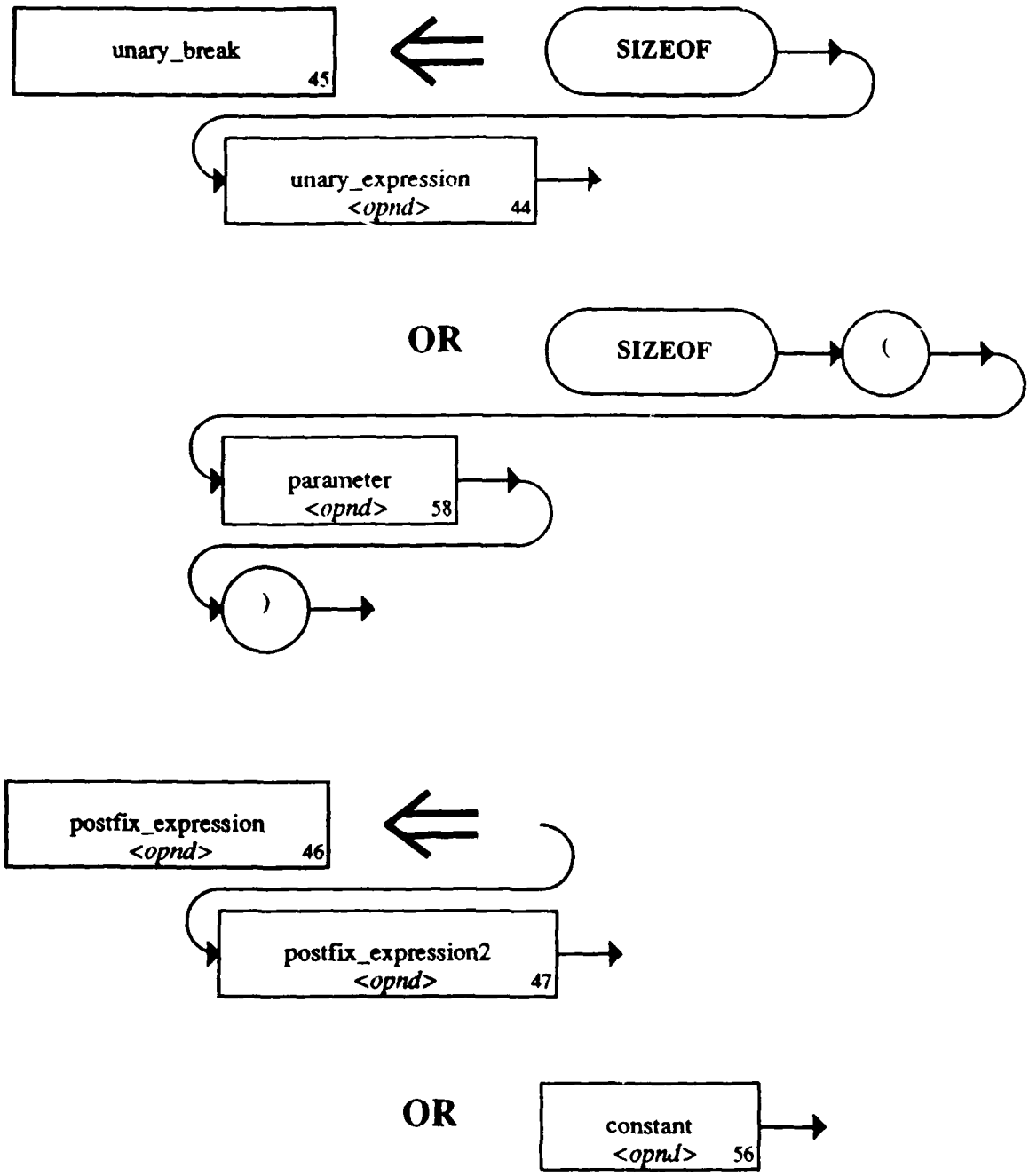
RAILROAD DIAGRAMS for DL GRAMMAR



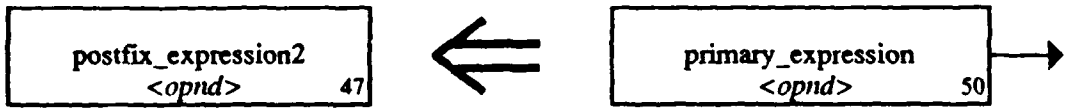
RAILROAD DIAGRAMS for DL GRAMMAR

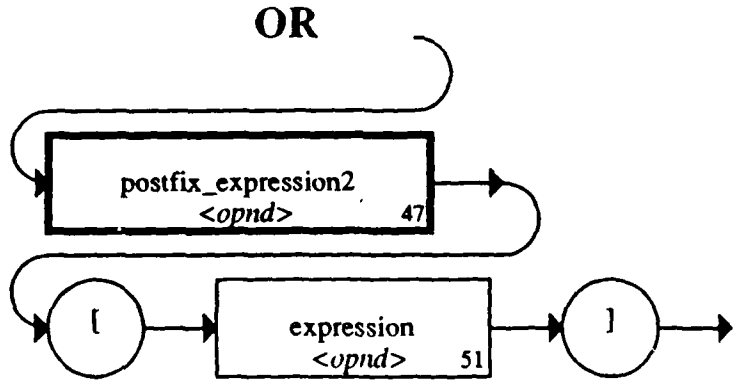


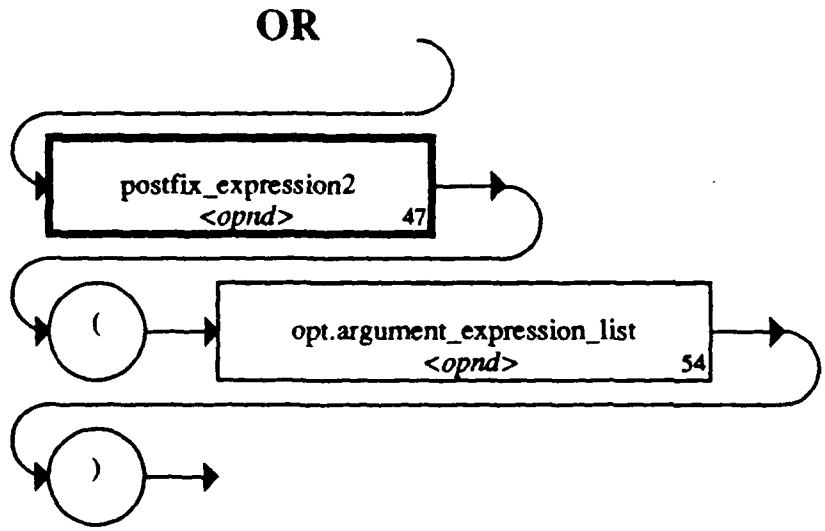
RAILROAD DIAGRAMS for DL GRAMMAR



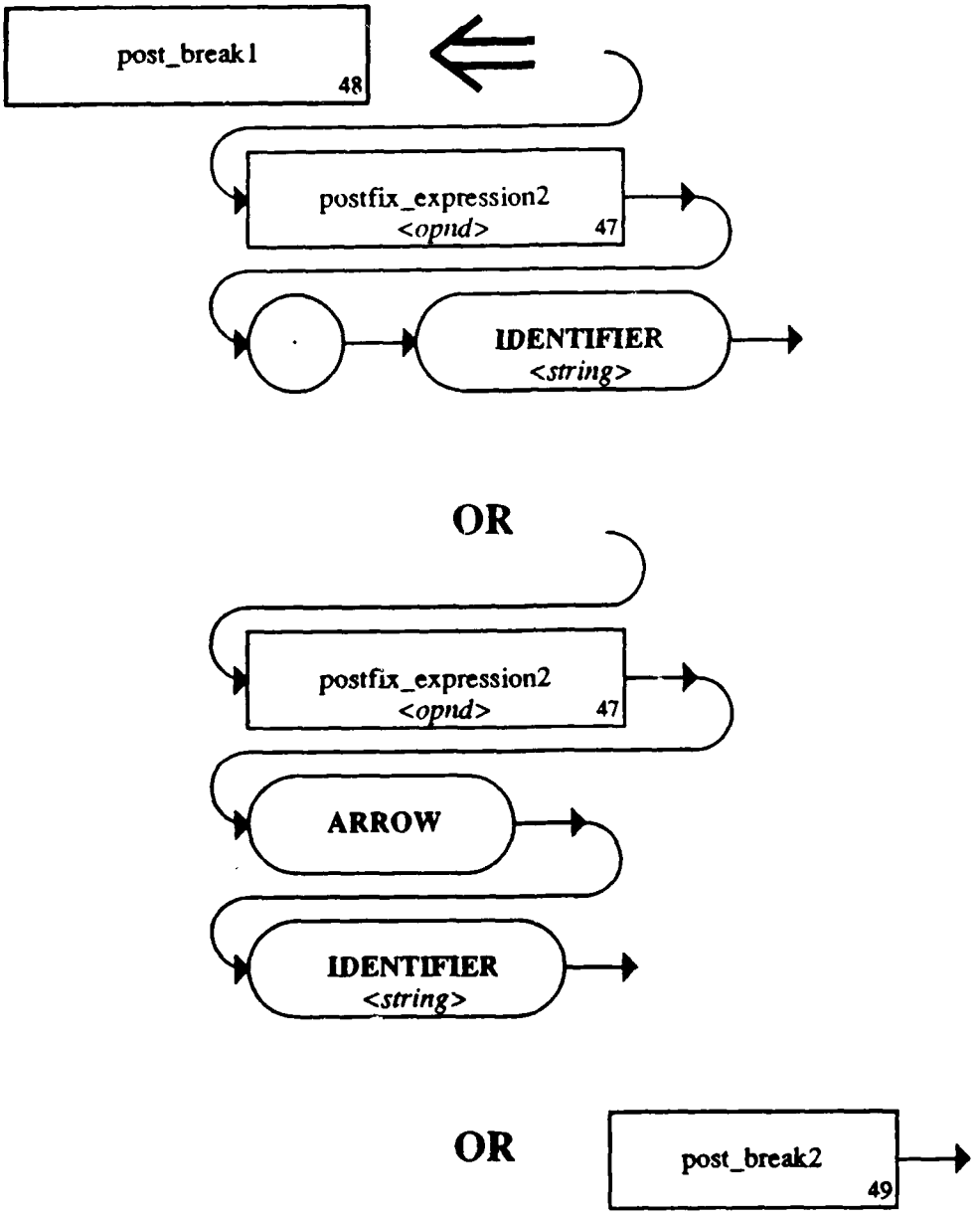
RAILROAD DIAGRAMS for DL GRAMMAR



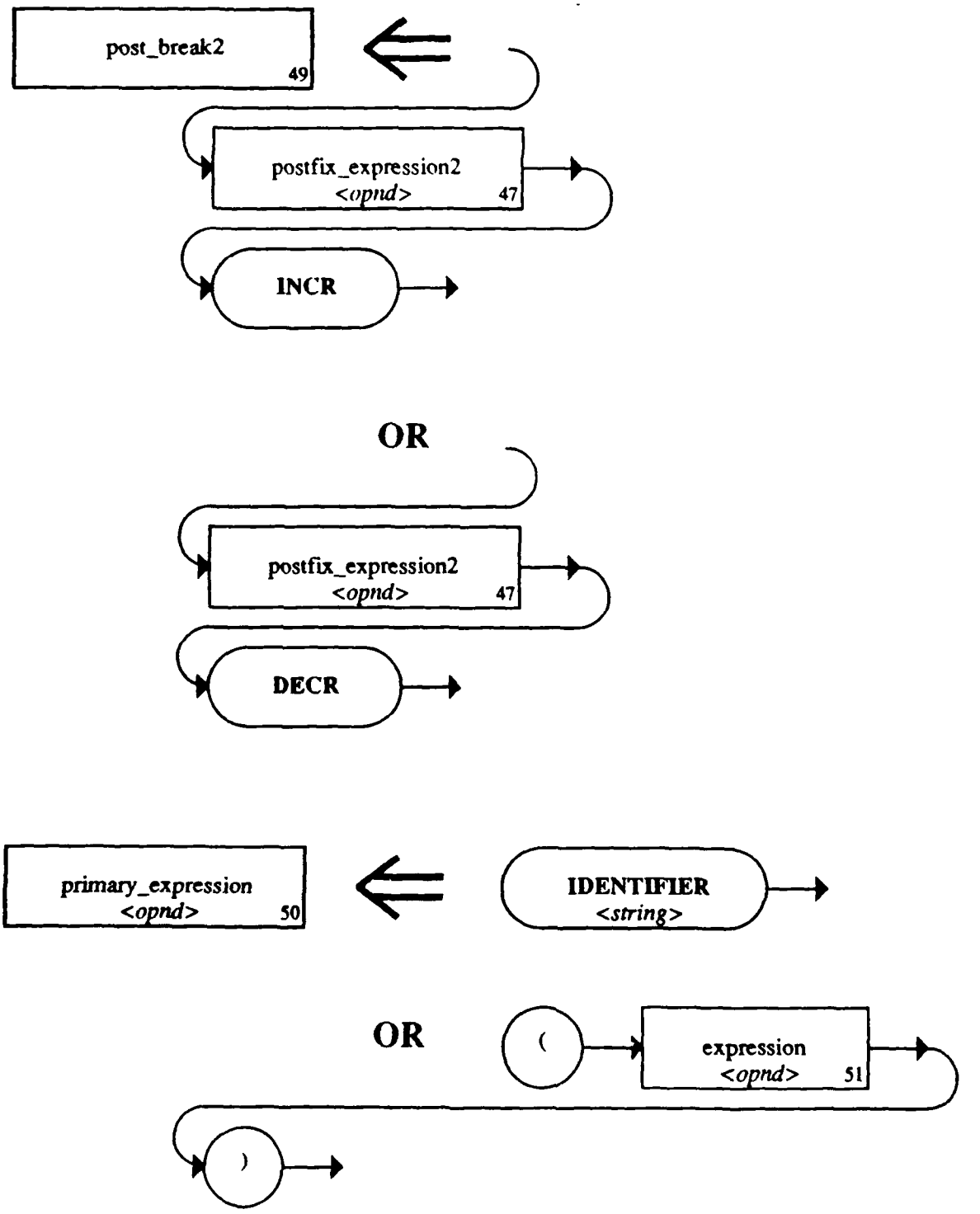




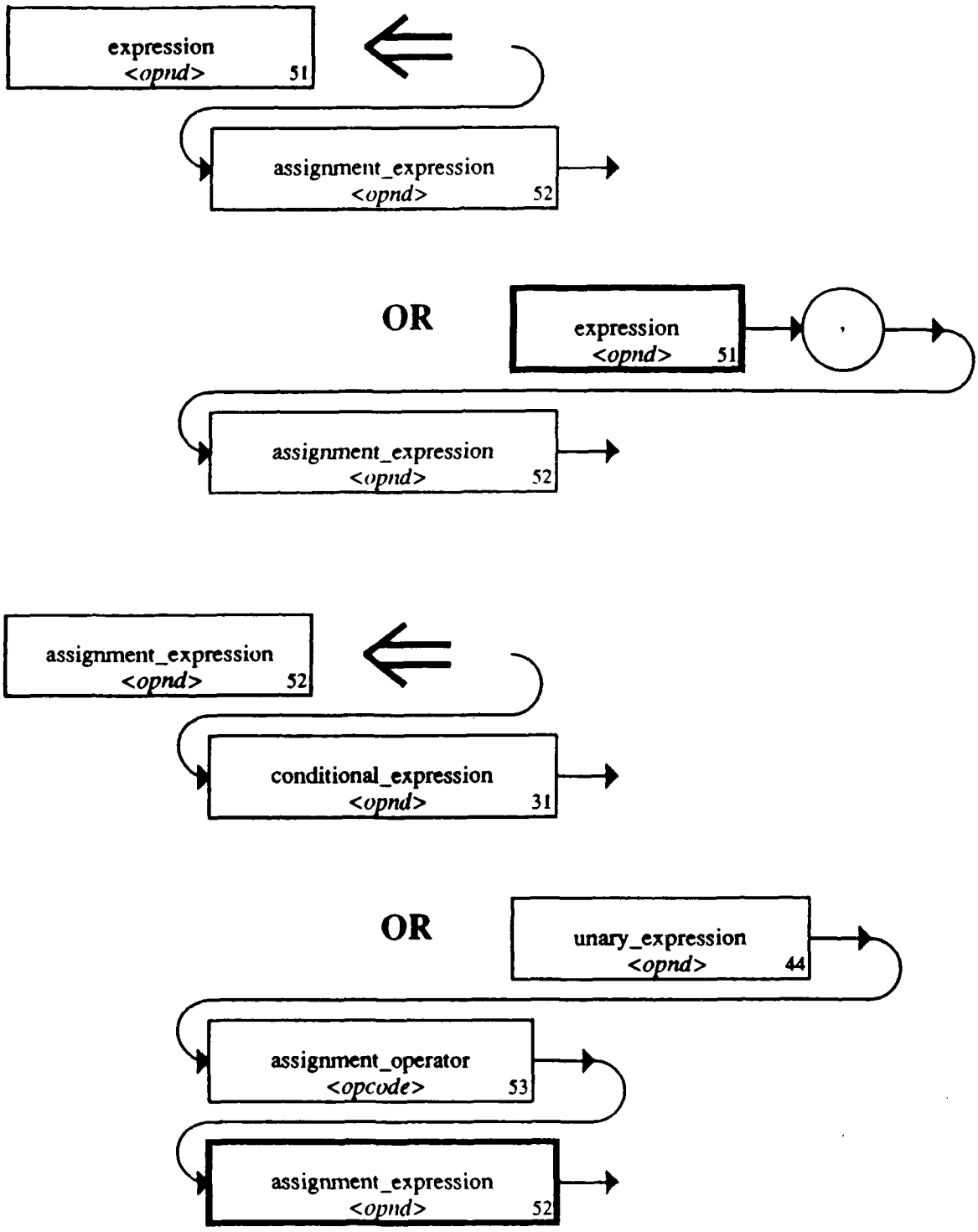
RAILROAD DIAGRAMS for DL GRAMMAR



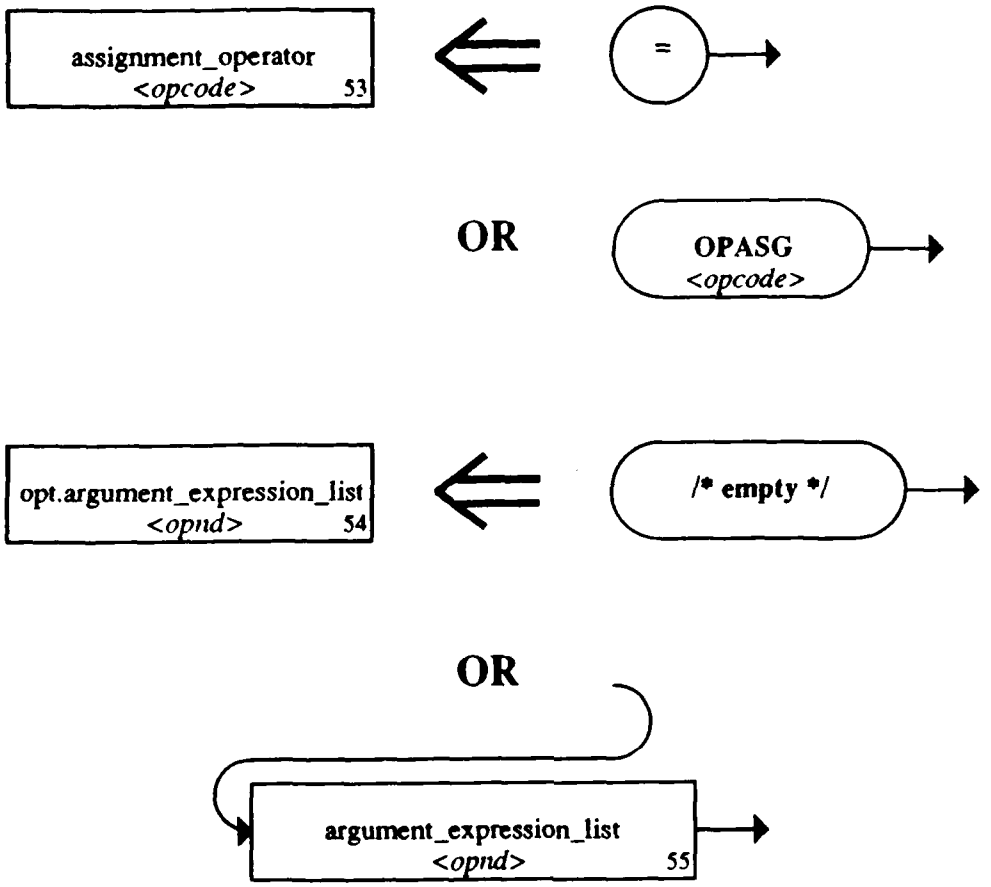
RAILROAD DIAGRAMS for DL GRAMMAR



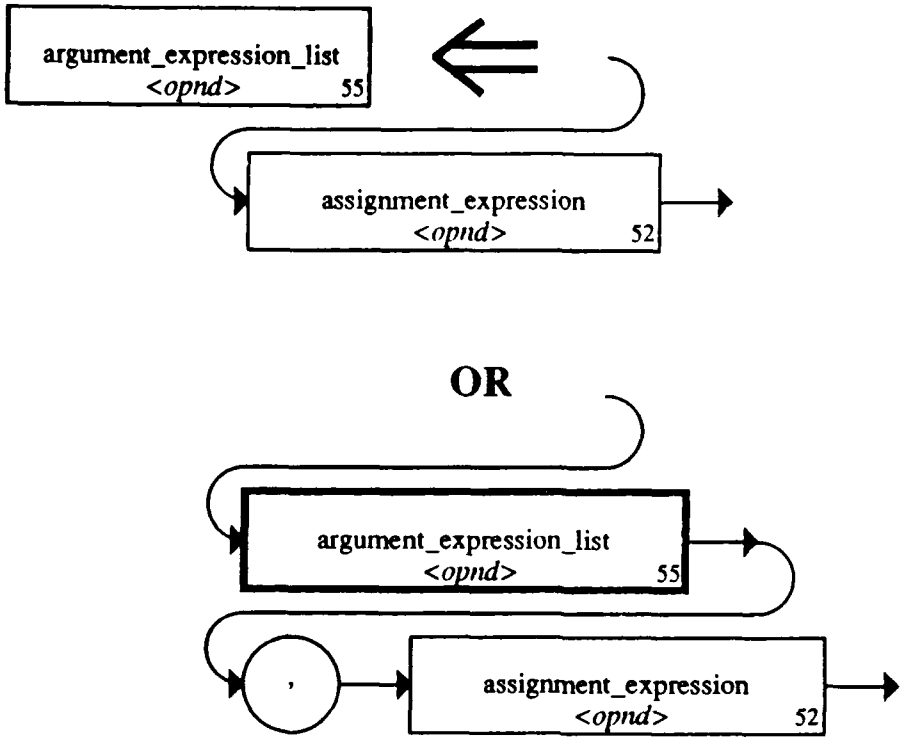
RAILROAD DIAGRAMS for DL GRAMMAR



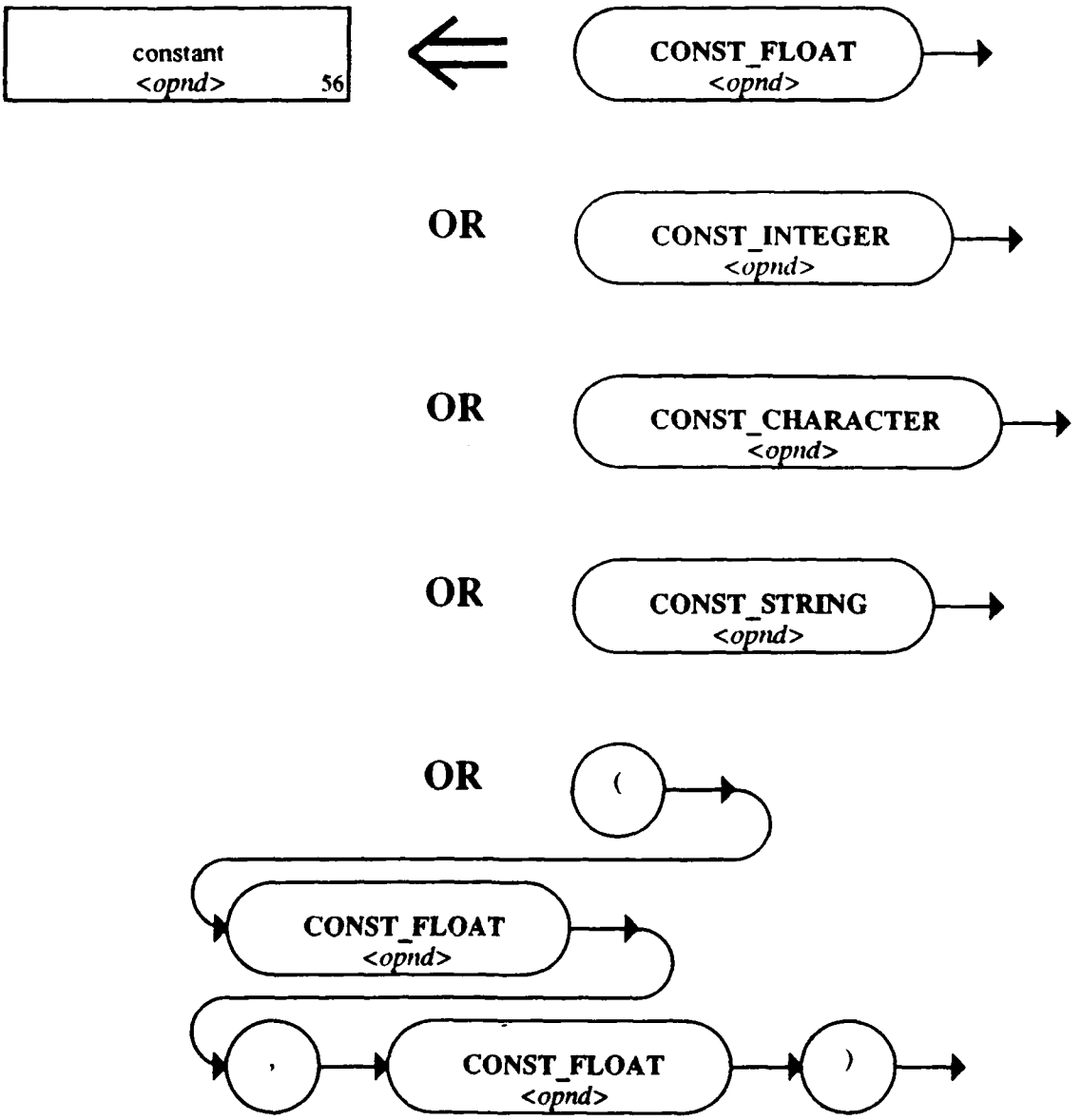
RAILROAD DIAGRAMS for DL GRAMMAR



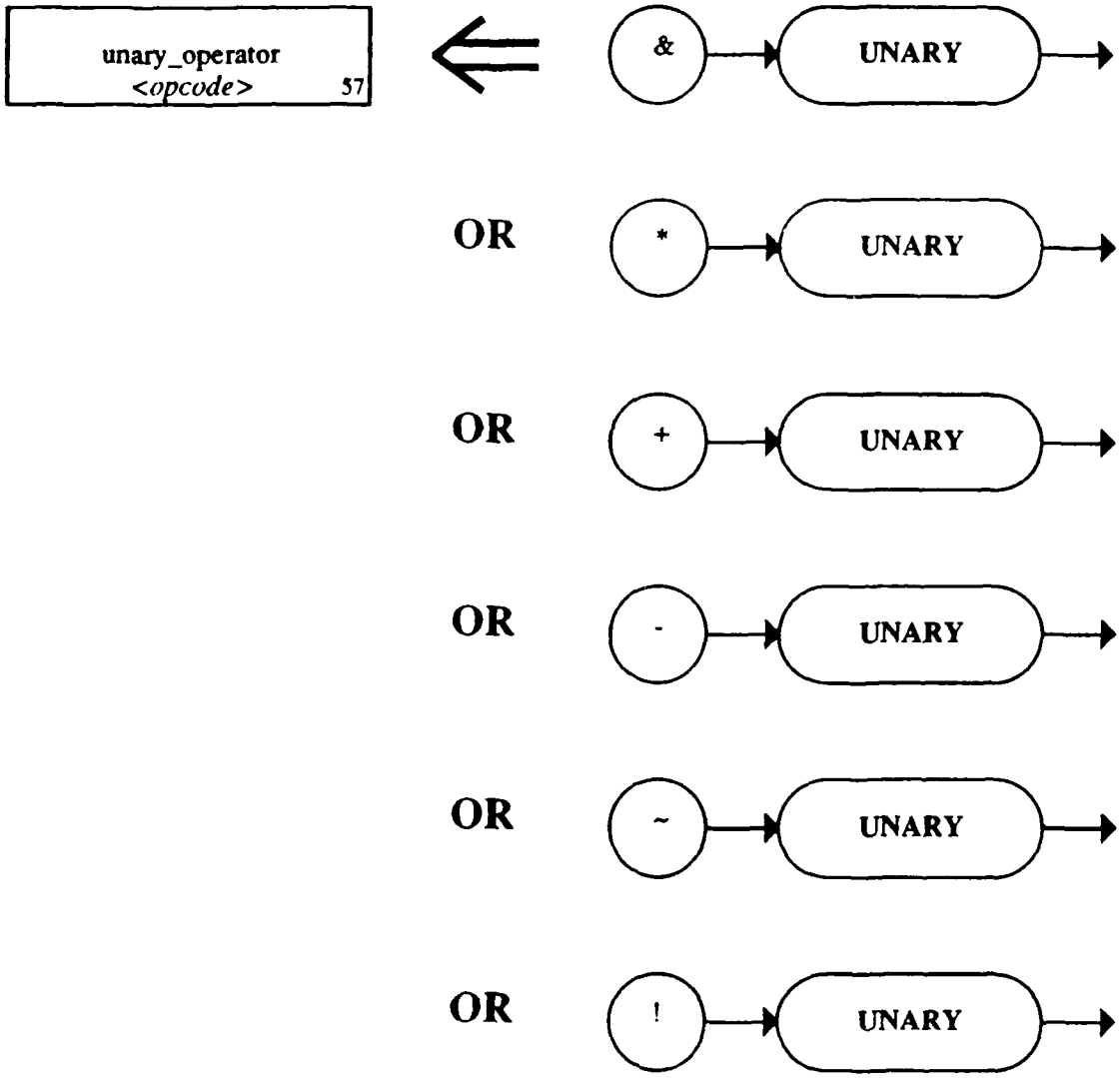
RAILROAD DIAGRAMS for DL GRAMMAR



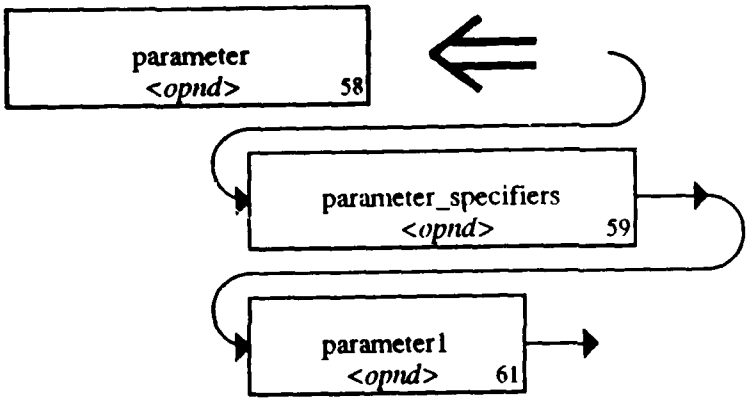
RAILROAD DIAGRAMS for DL GRAMMAR



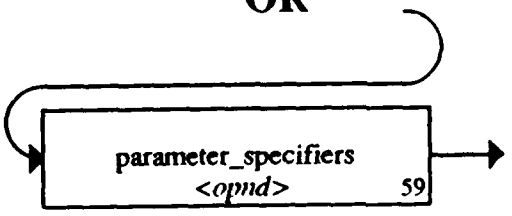
RAILROAD DIAGRAMS for DL GRAMMAR

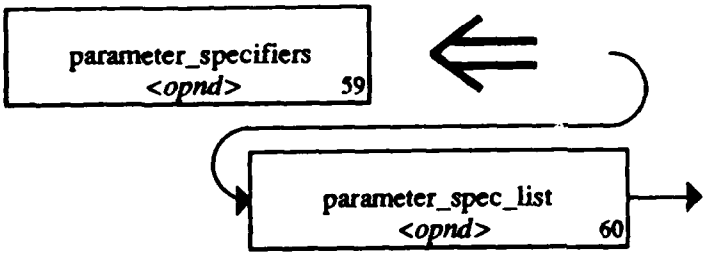


RAILROAD DIAGRAMS for DL GRAMMAR

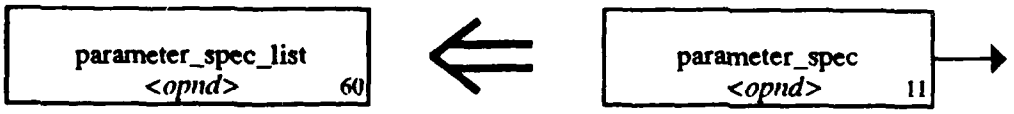


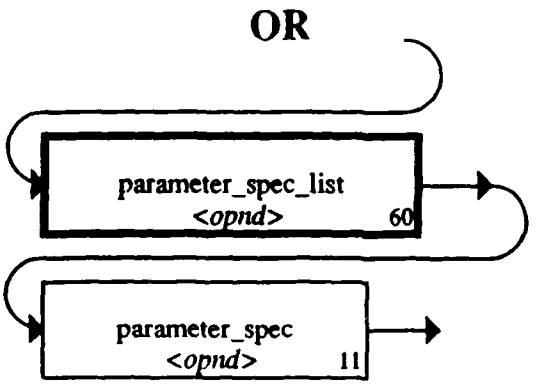
OR

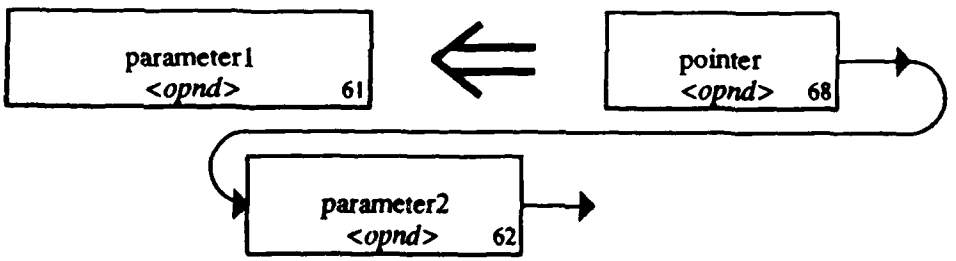


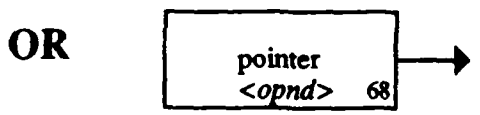


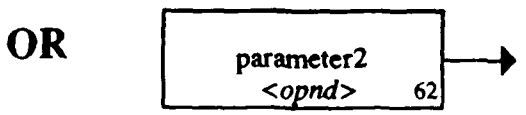
RAILROAD DIAGRAMS for DL GRAMMAR



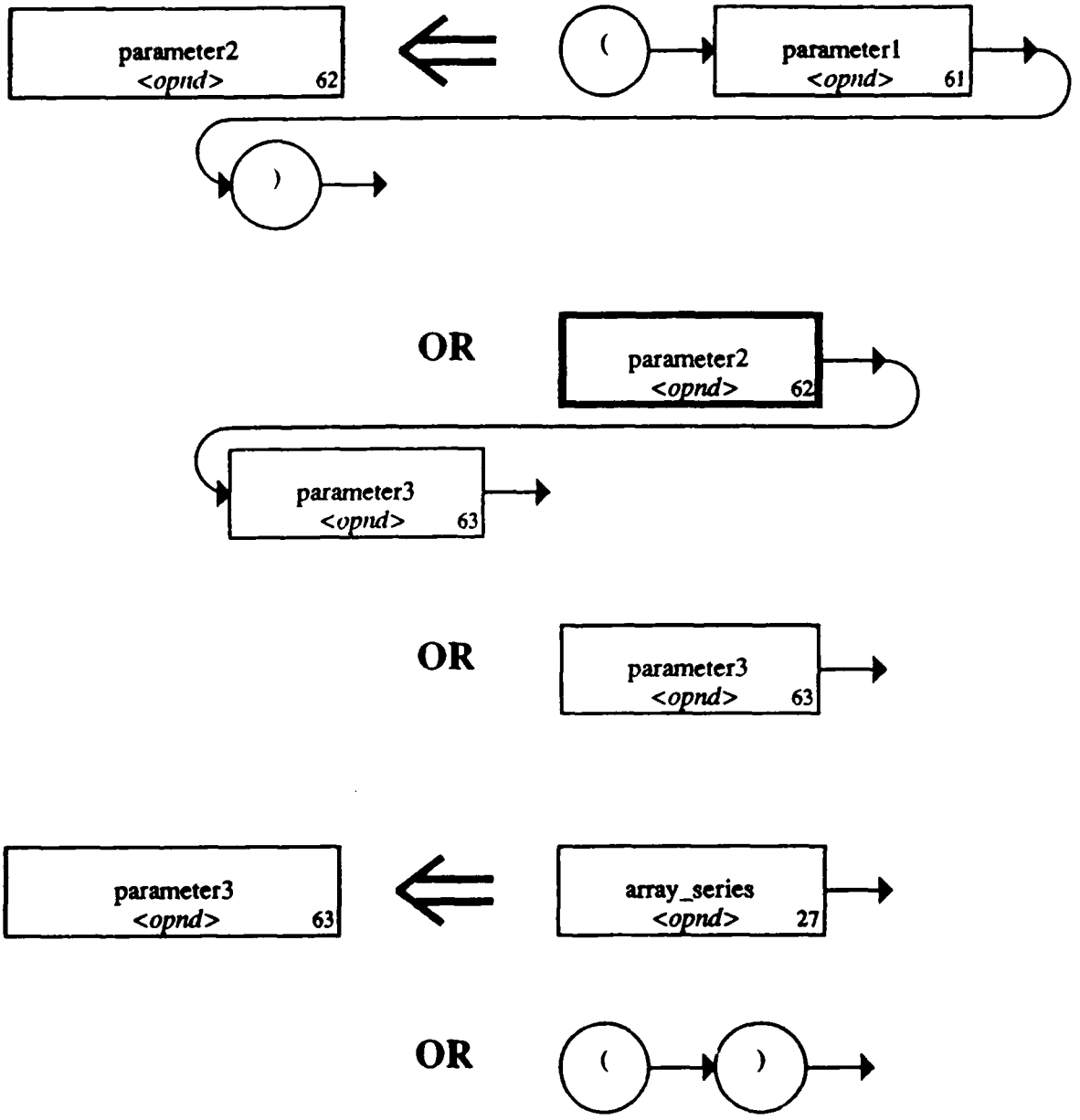




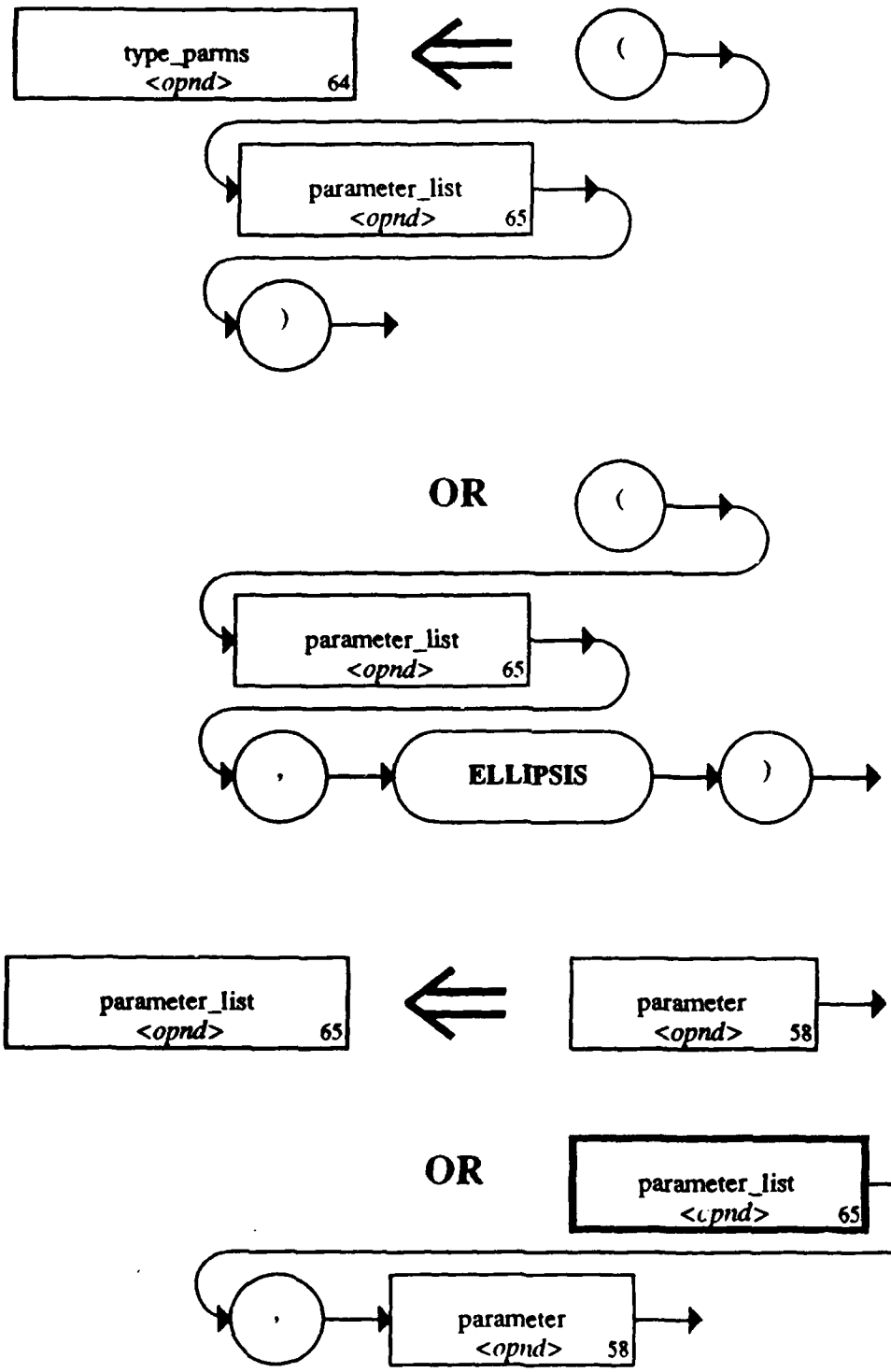




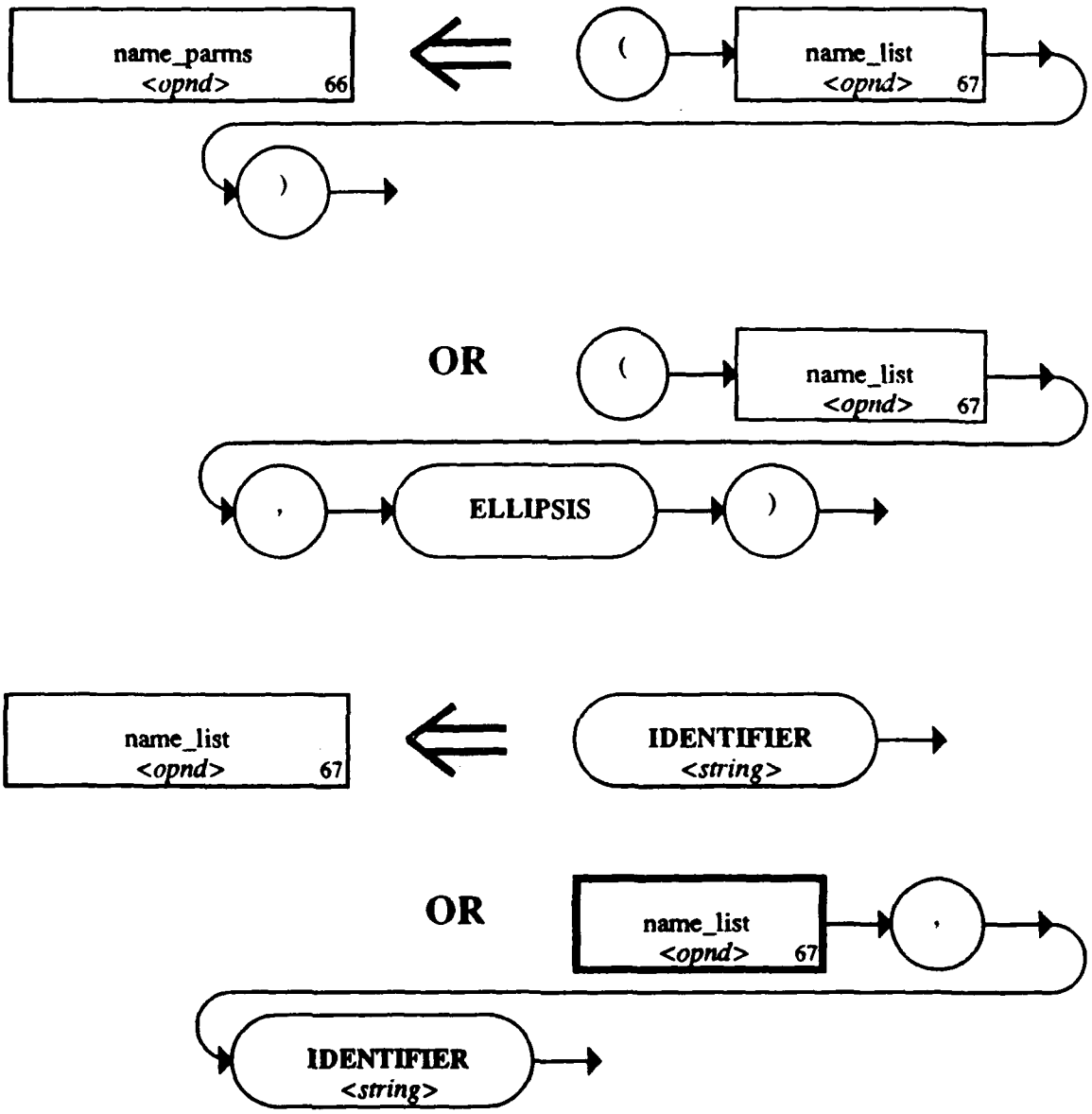
RAILROAD DIAGRAMS for DL GRAMMAR



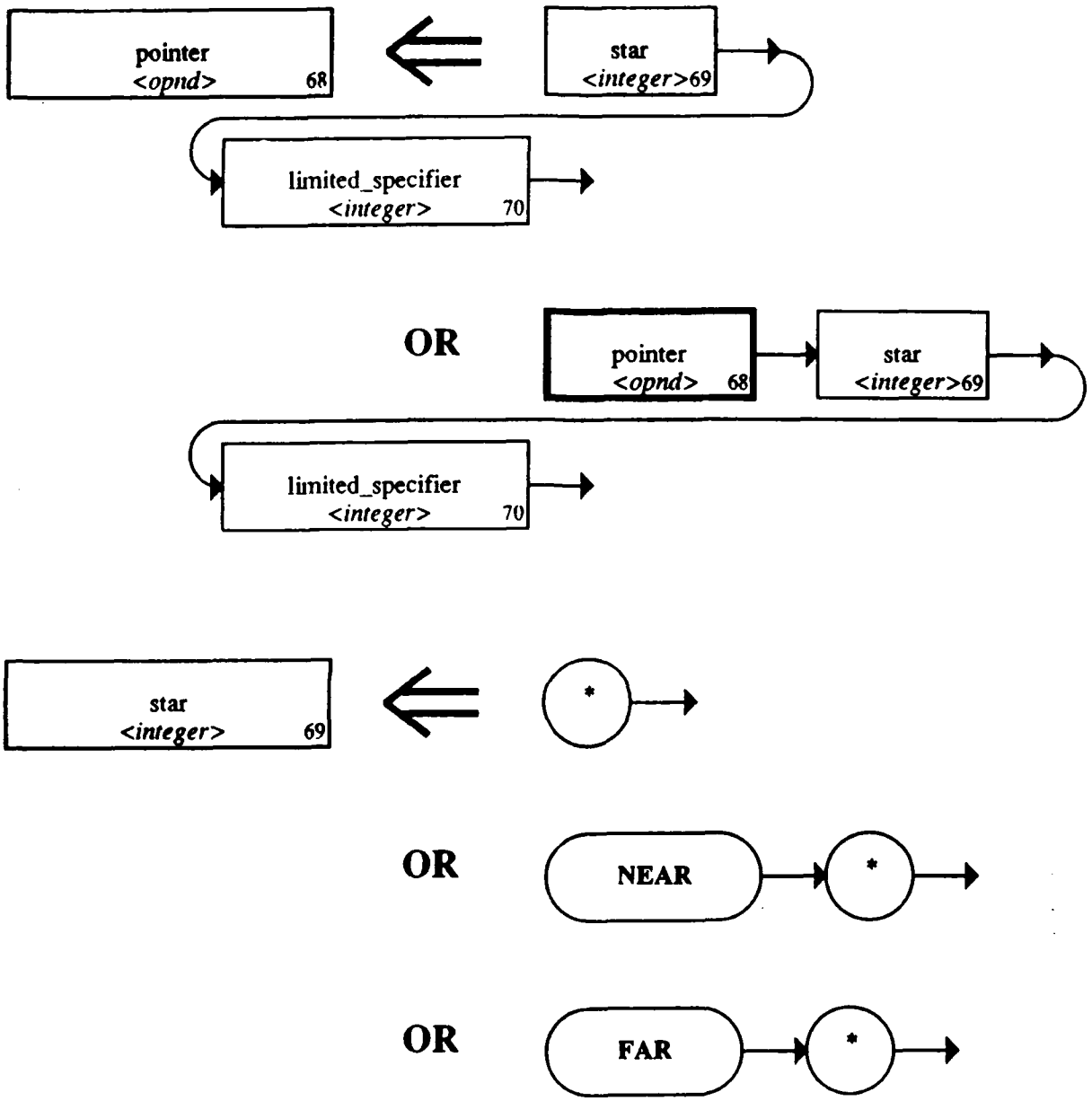
RAILROAD DIAGRAMS for DL GRAMMAR



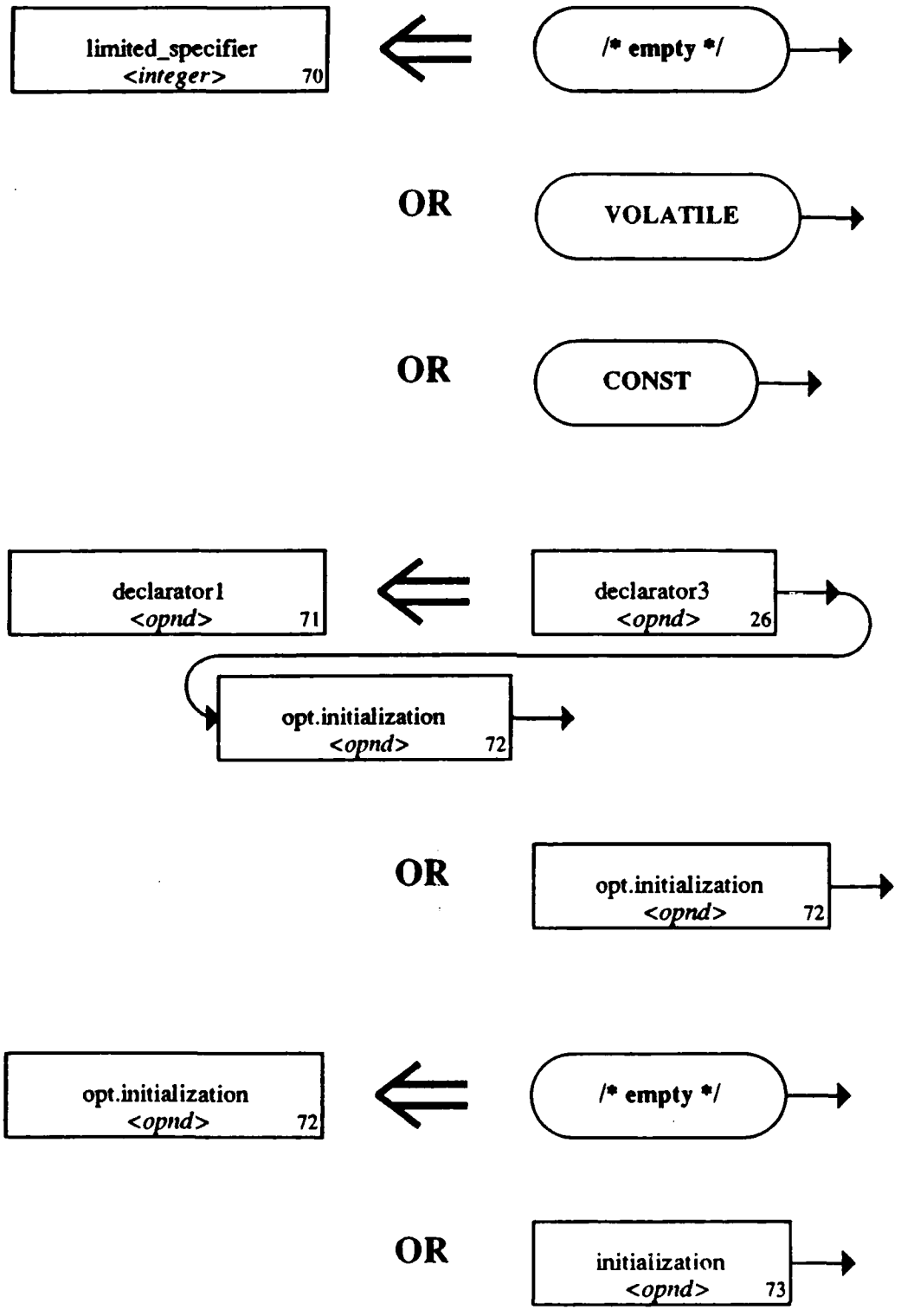
RAILROAD DIAGRAMS for DL GRAMMAR



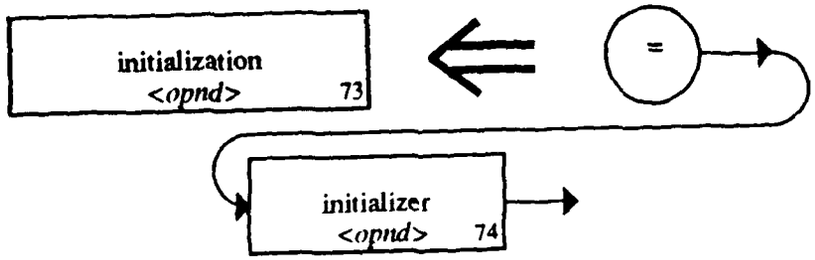
RAILROAD DIAGRAMS for DL GRAMMAR



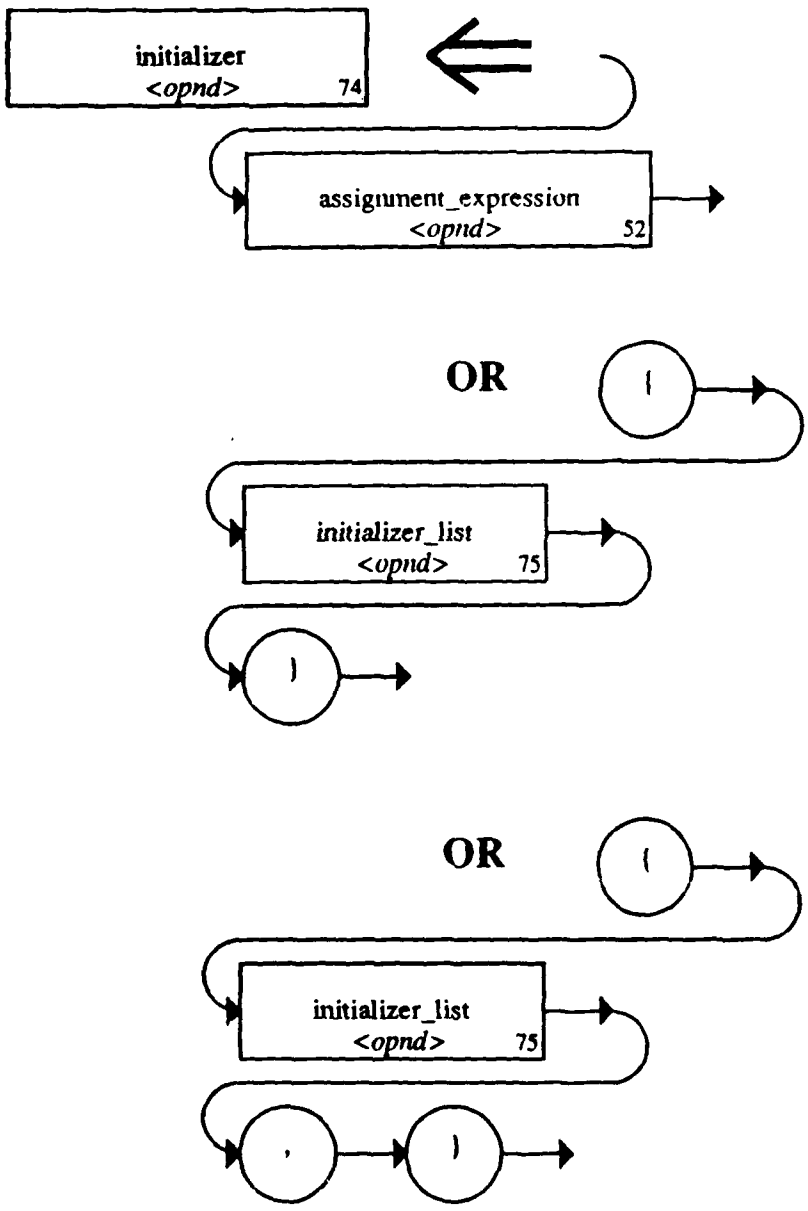
RAILROAD DIAGRAMS for DL GRAMMAR



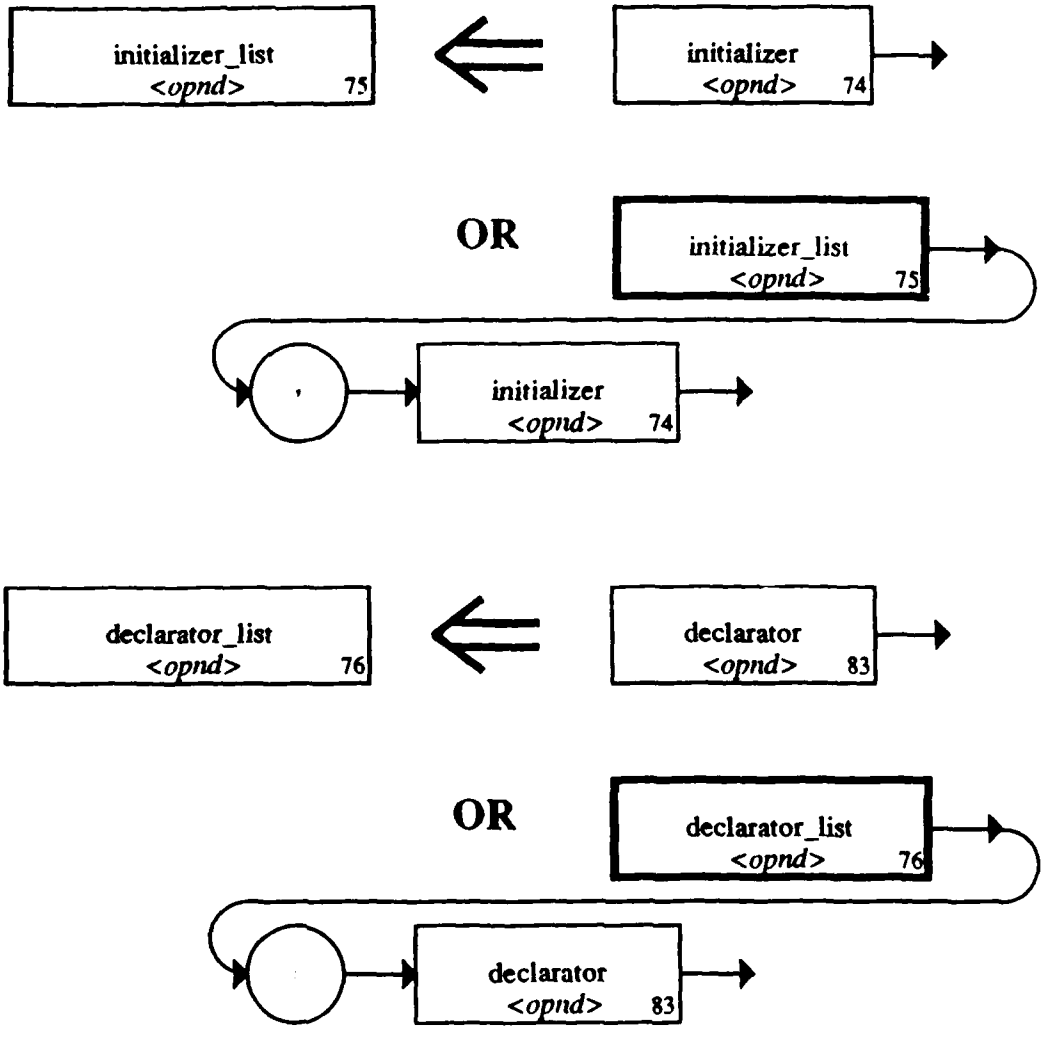
RAILROAD DIAGRAMS for DL GRAMMAR



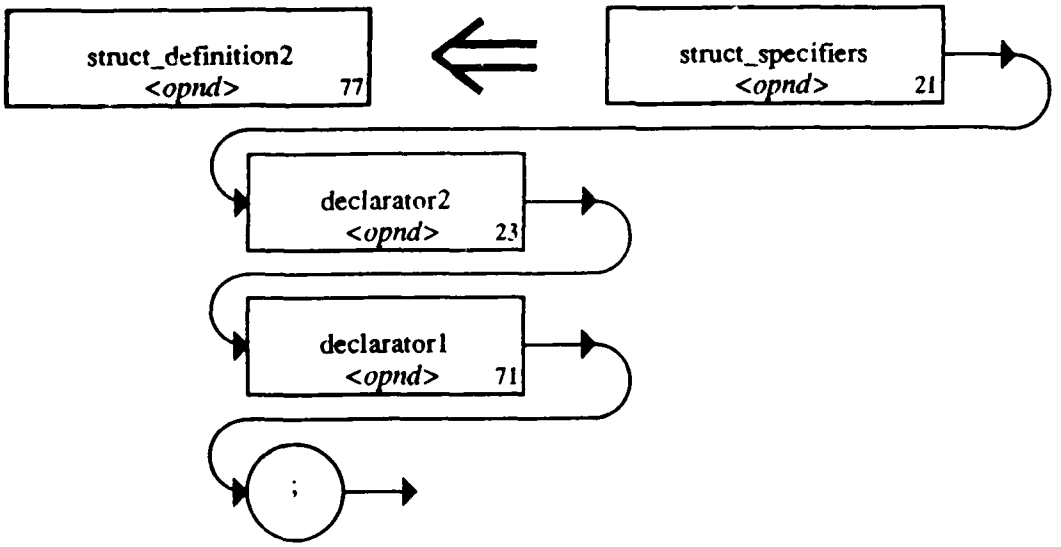
RAILROAD DIAGRAMS for DL GRAMMAR

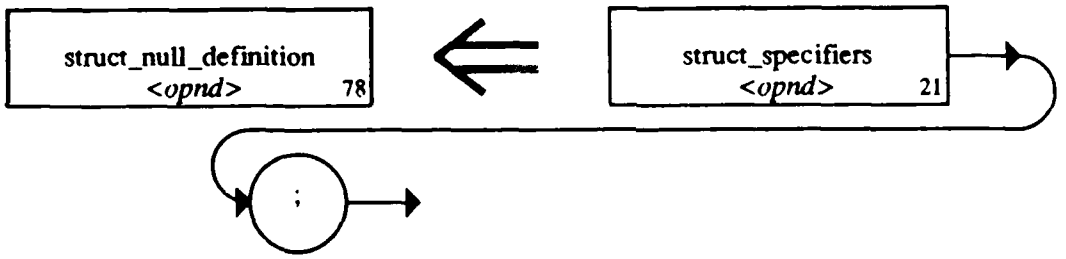


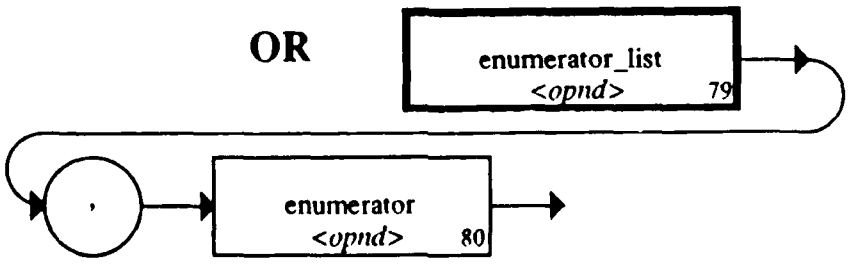
RAILROAD DIAGRAMS for DL GRAMMAR



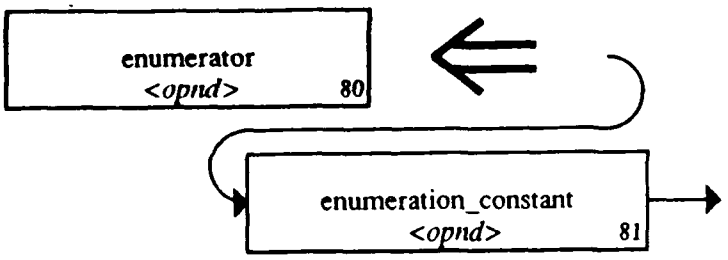
RAILROAD DIAGRAMS for DL GRAMMAR



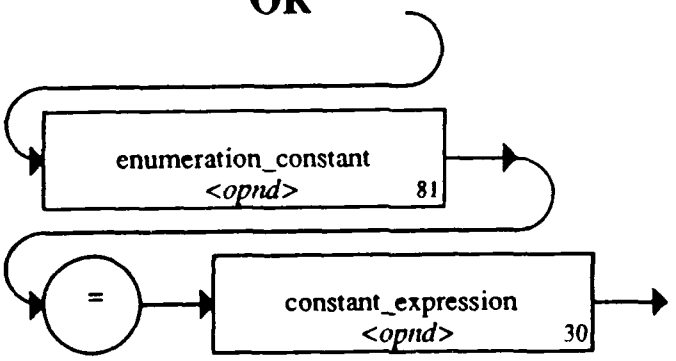




RAILROAD DIAGRAMS for DL GRAMMAR

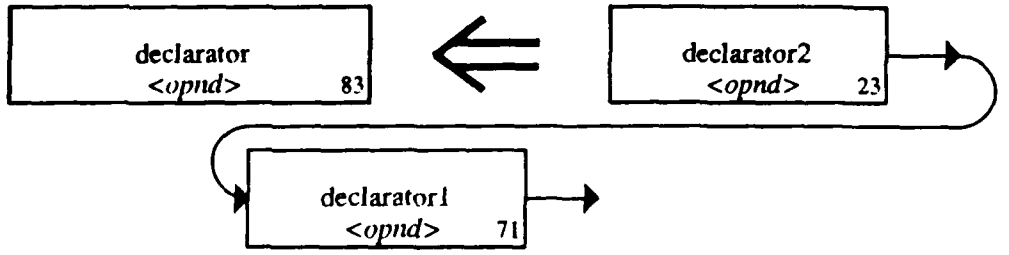


OR

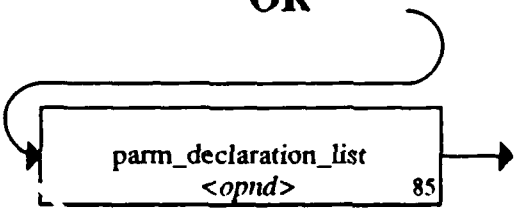


OR

RAILROAD DIAGRAMS for DL GRAMMAR

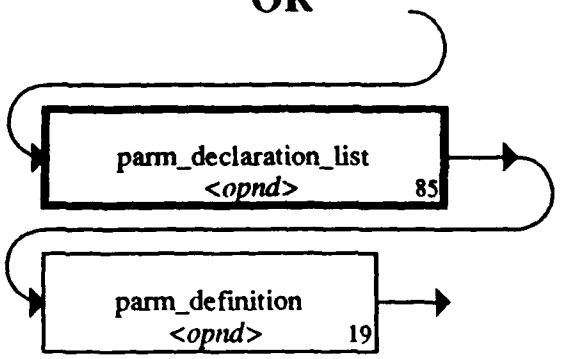


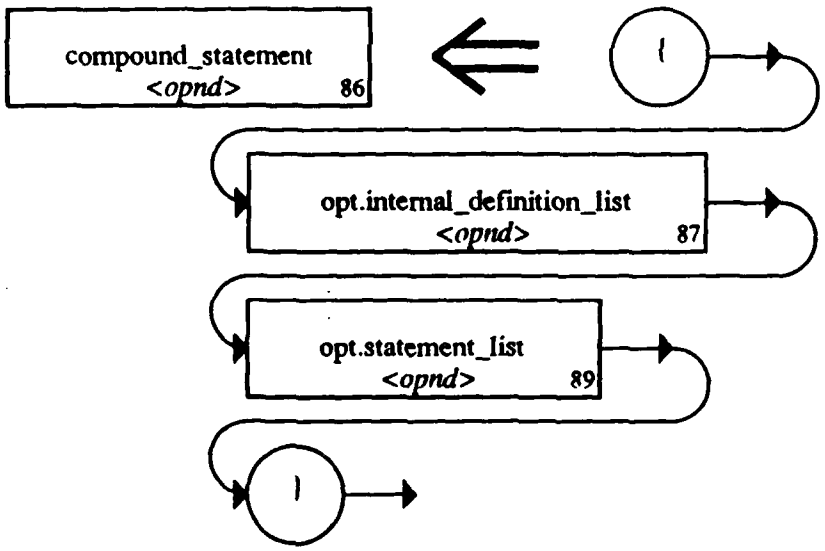
OR



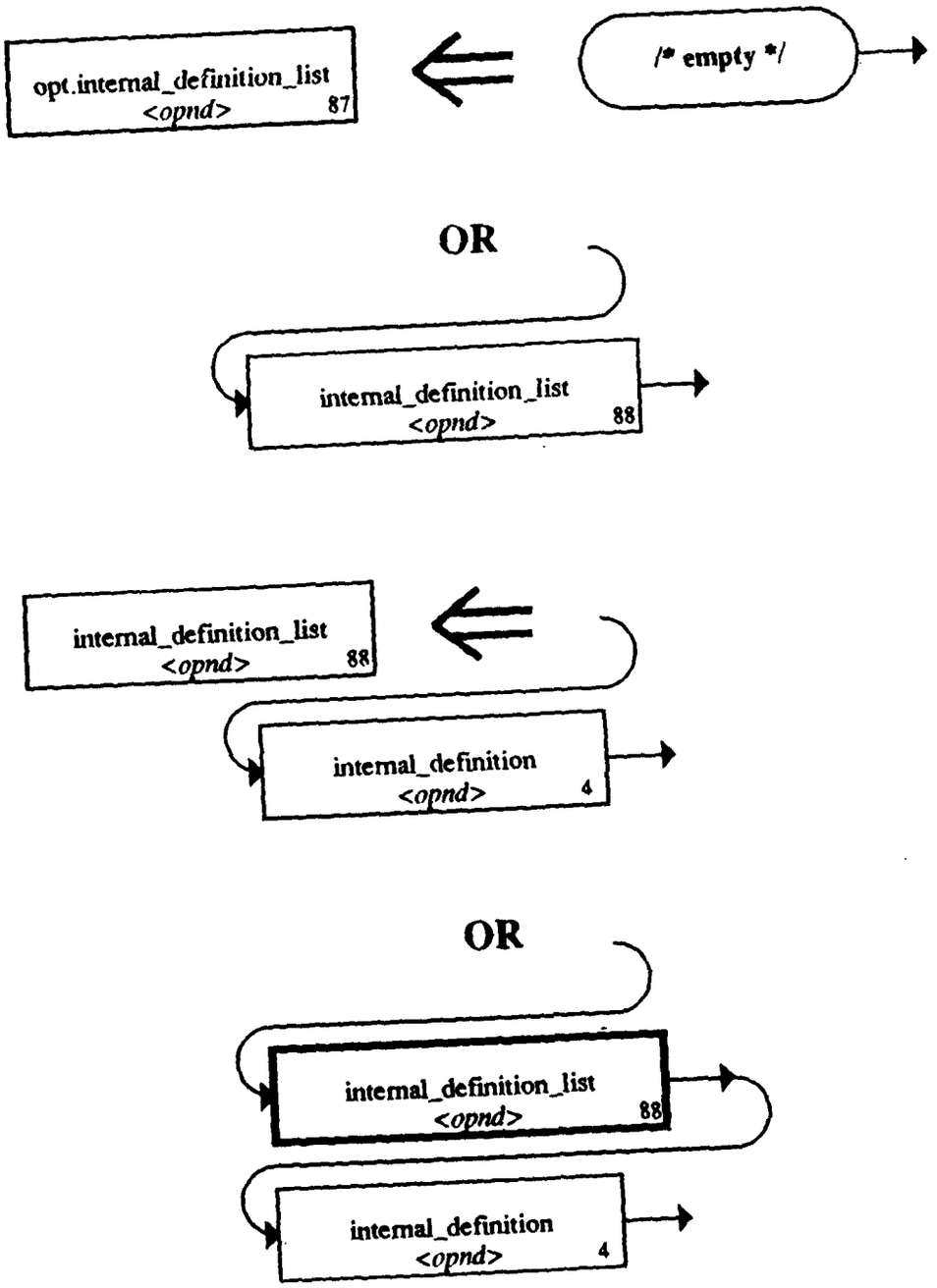
RAILROAD DIAGRAMS for DL GRAMMAR

OR

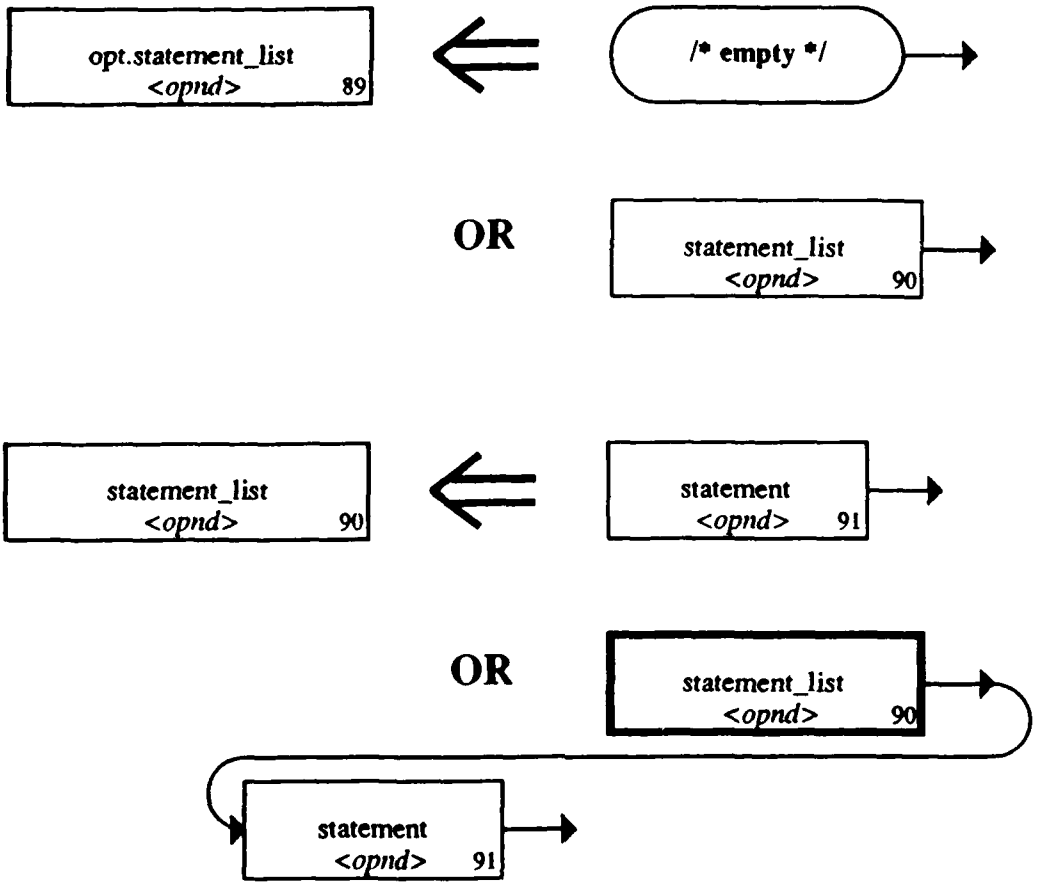




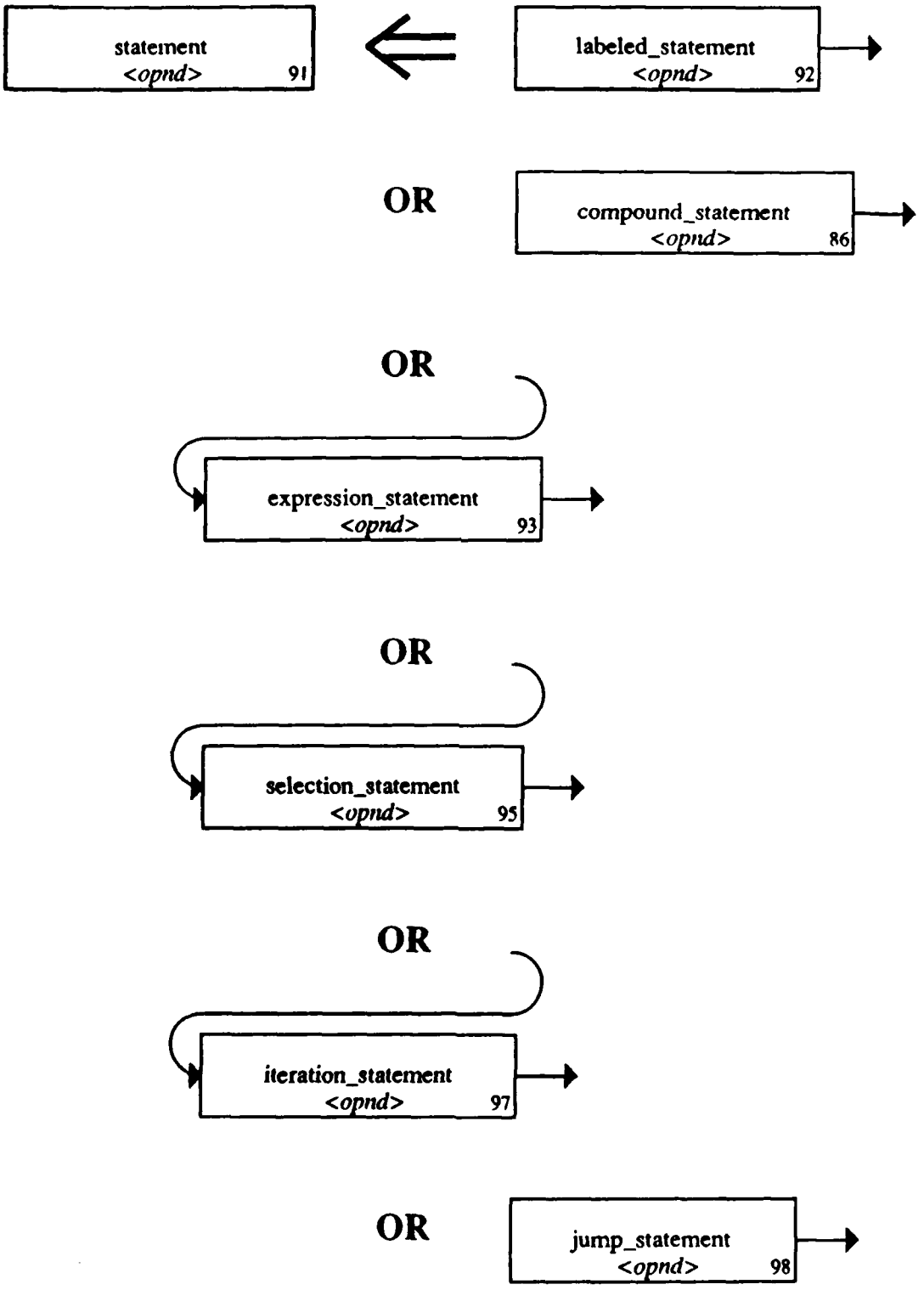
RAILROAD DIAGRAMS for DL GRAMMAR



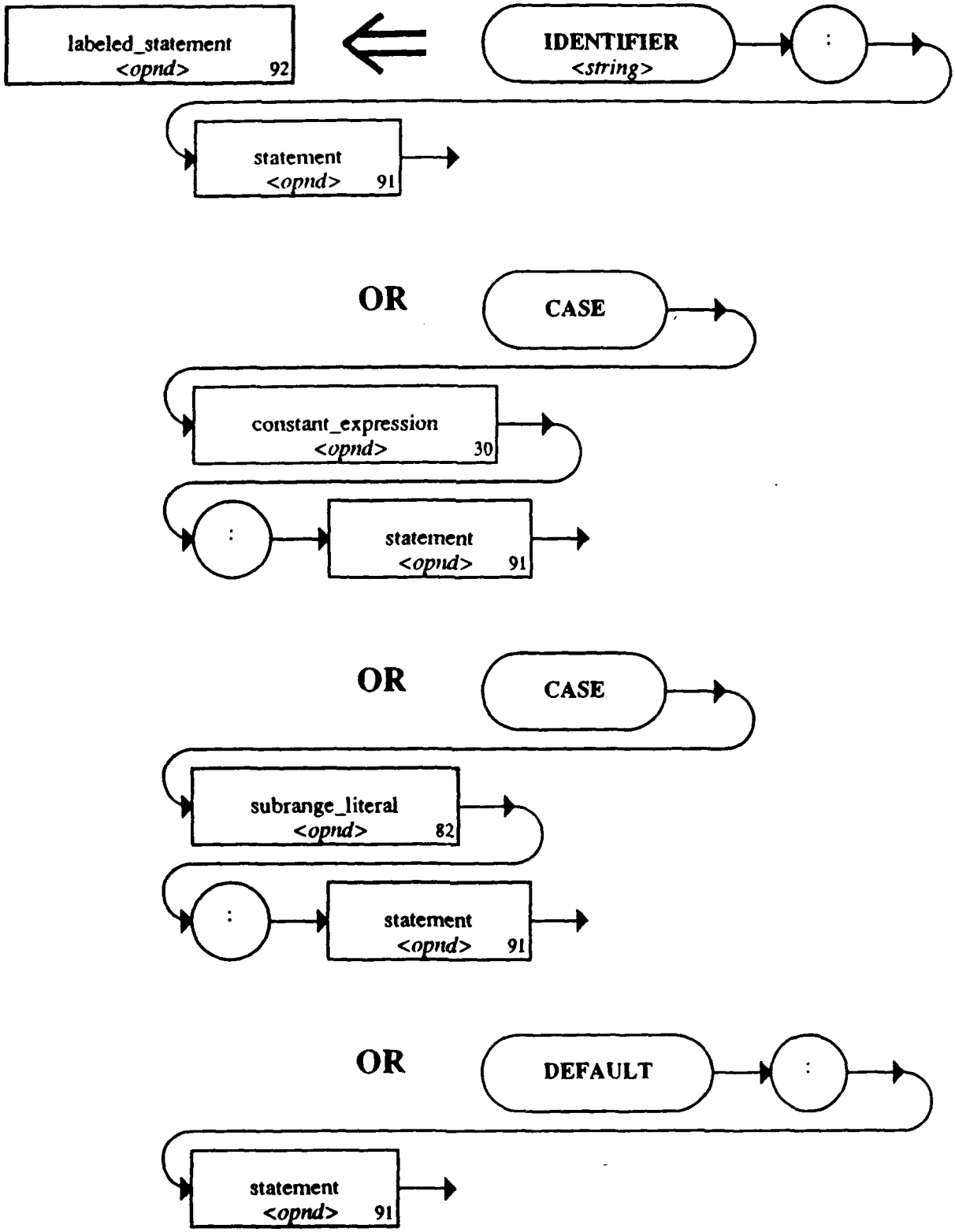
RAILROAD DIAGRAMS for DL GRAMMAR



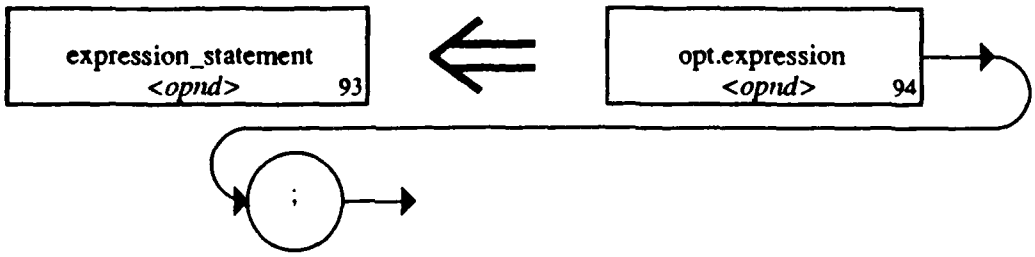
RAILROAD DIAGRAMS for DL GRAMMAR



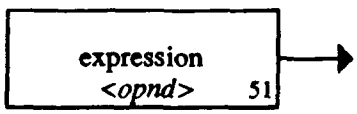
RAILROAD DIAGRAMS for DL GRAMMAR



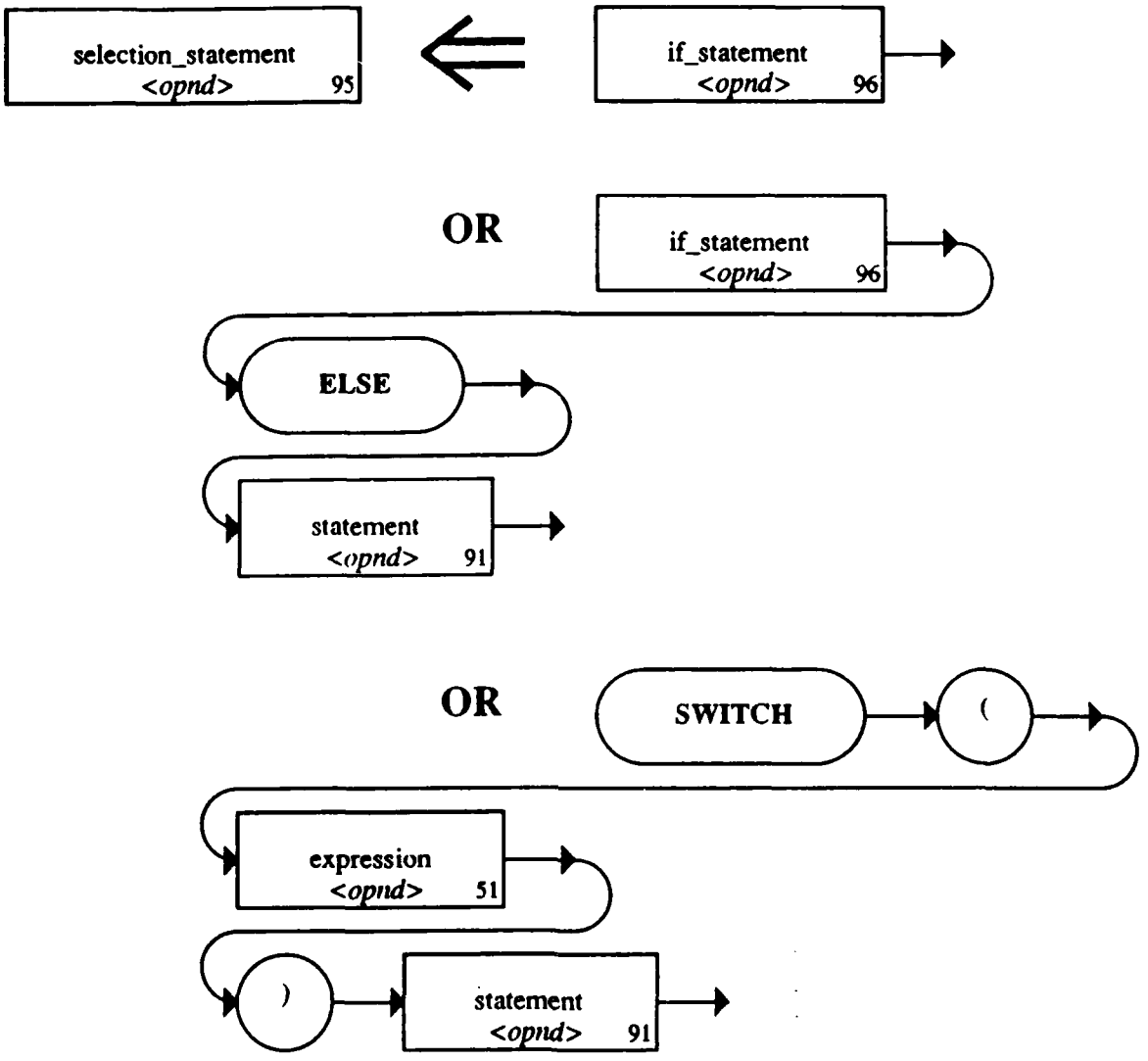
RAILROAD DIAGRAMS for DL GRAMMAR



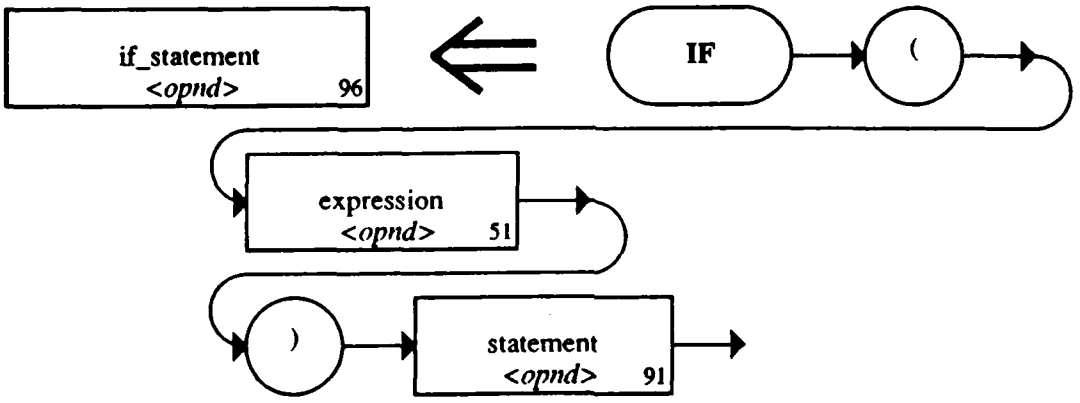
OR



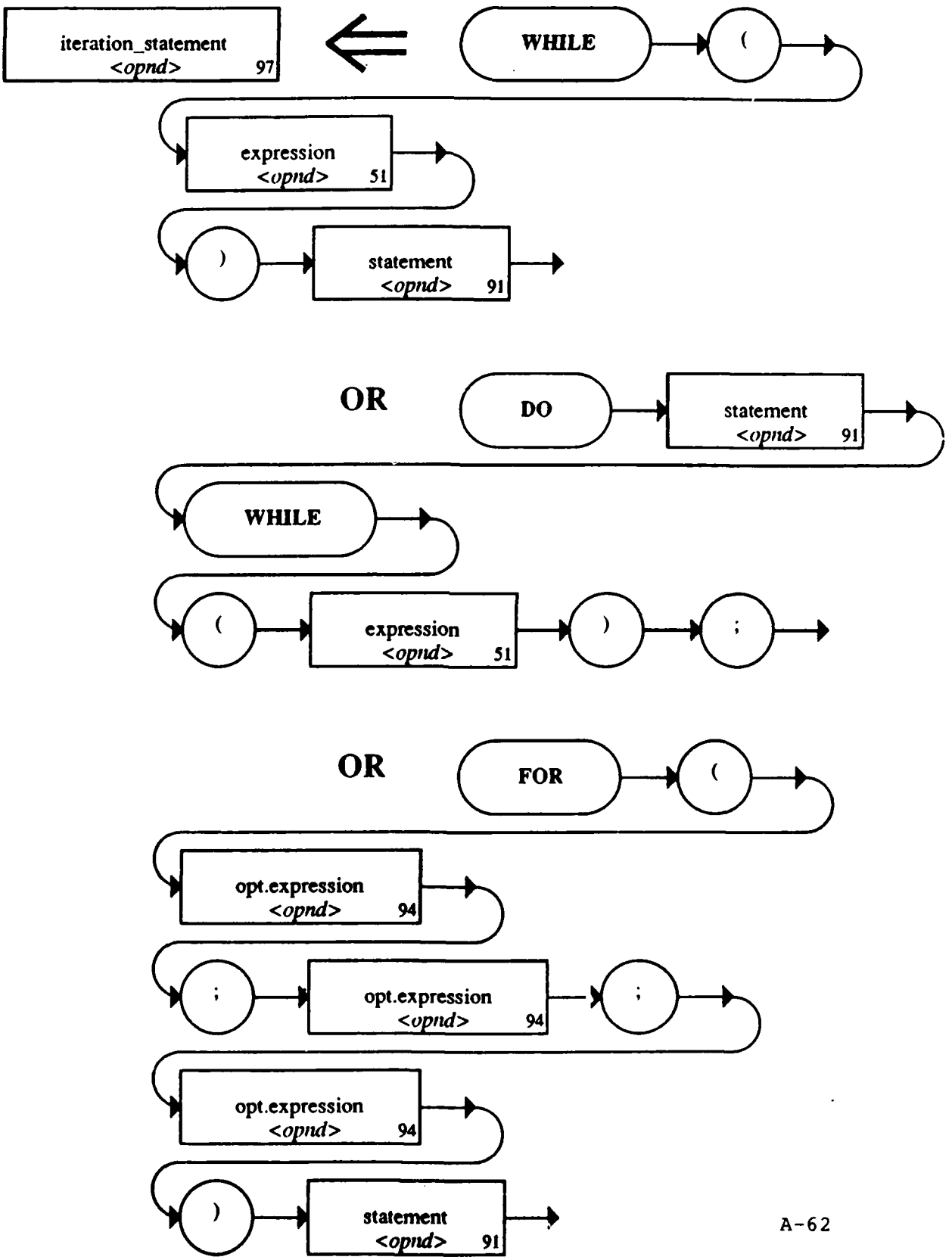
RAILROAD DIAGRAMS for DL GRAMMAR



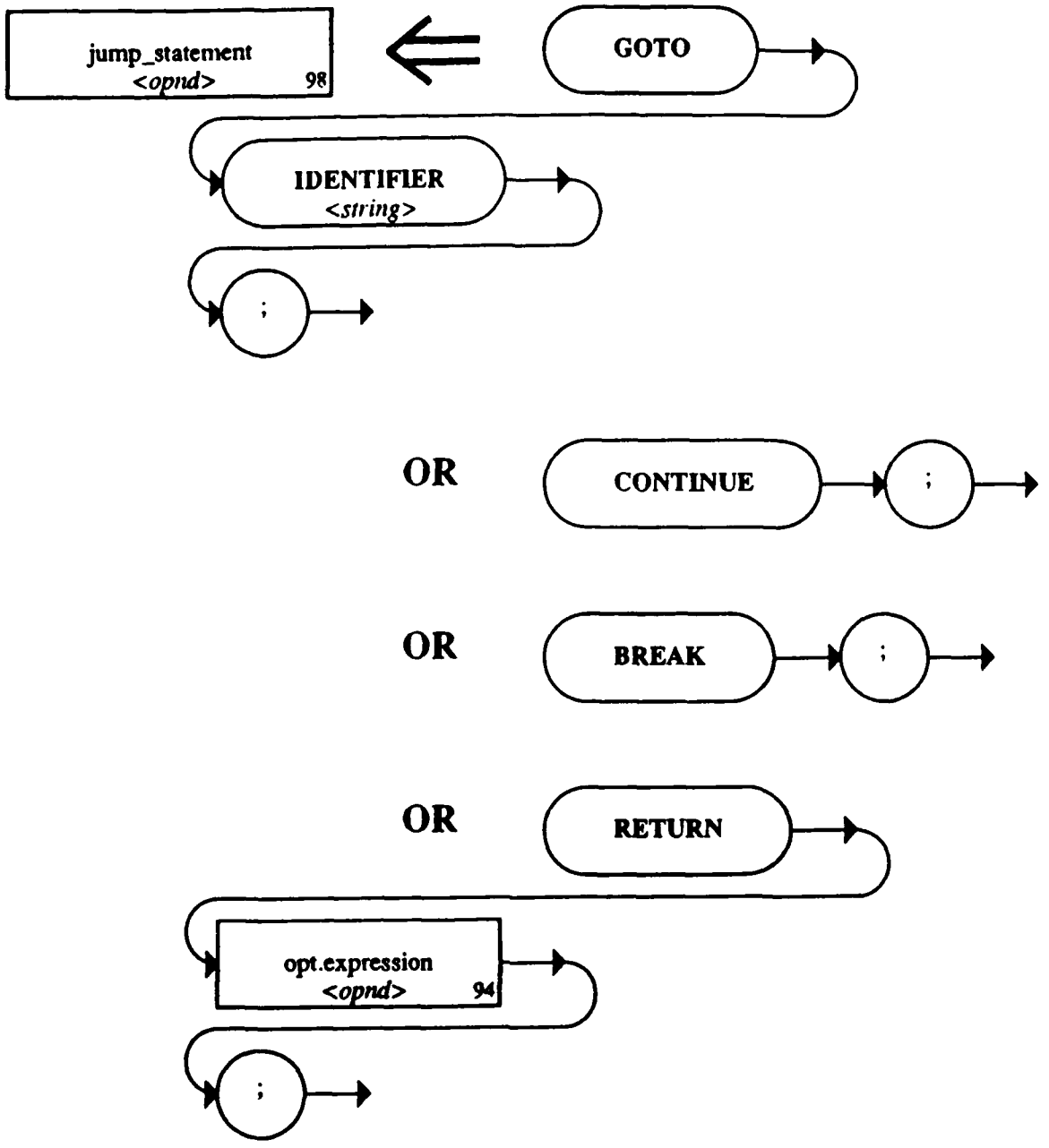
RAILROAD DIAGRAMS for DL GRAMMAR



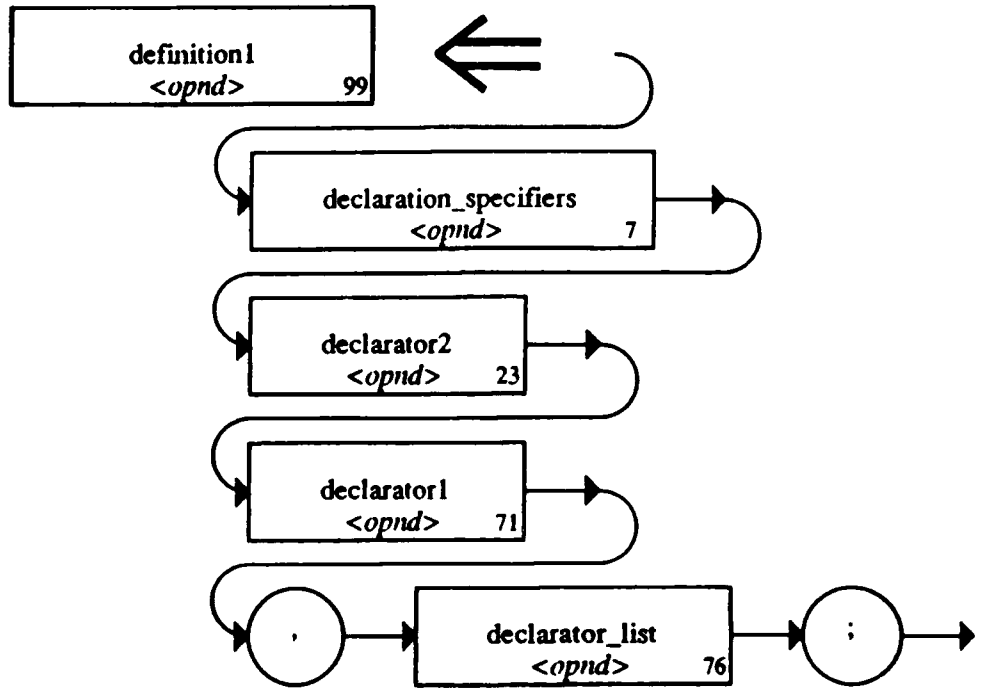
RAILROAD DIAGRAMS for DL GRAMMAR

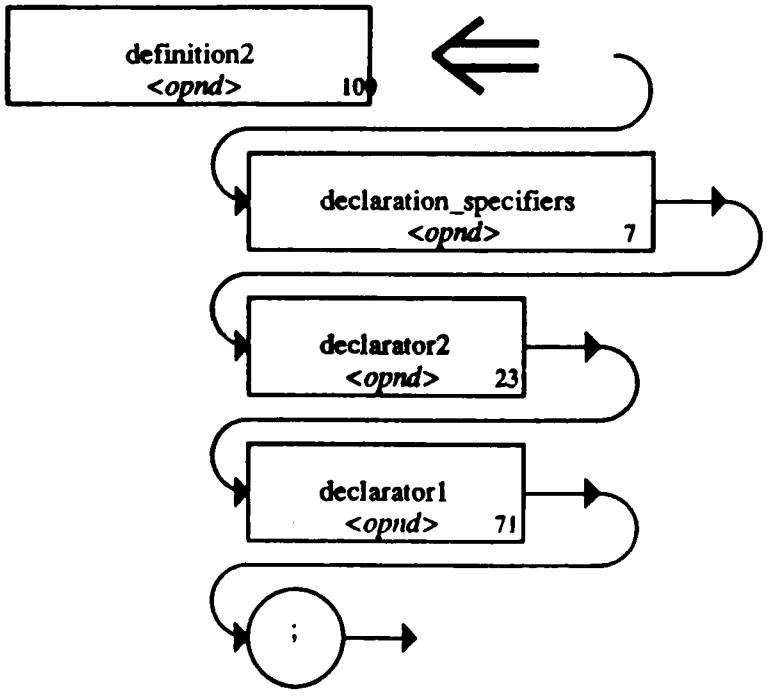


RAILROAD DIAGRAMS for DL GRAMMAR

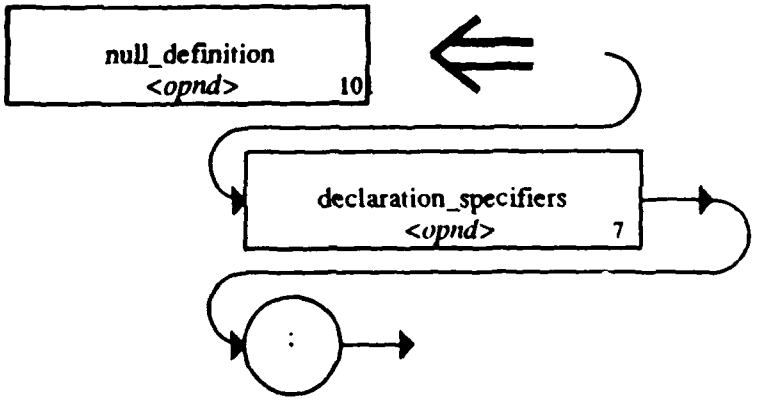


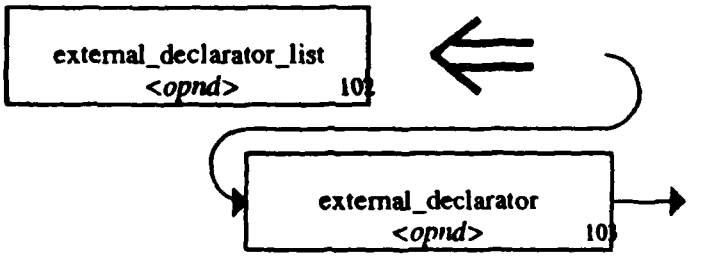
RAILROAD DIAGRAMS for DL GRAMMAR



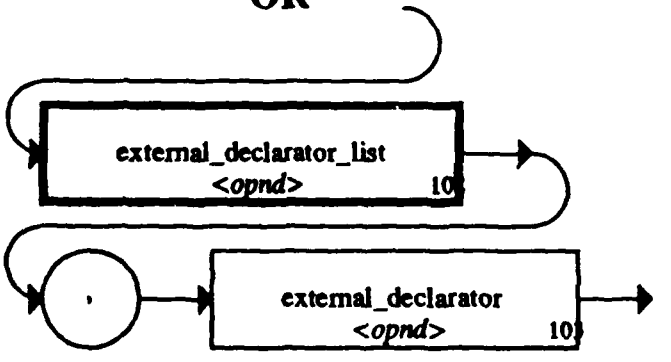


RAILROAD DIAGRAMS for DL GRAMMAR

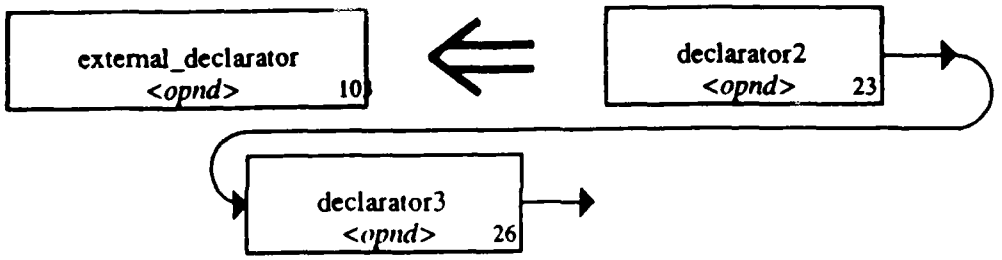




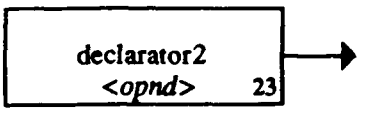
OR



RAILROAD DIAGRAMS for DL GRAMMAR



OR



APPENDIX B

IATHRL

AN IMAGE THRESHOLDING ROUTINE

THE C-CODED ROUTINE

```
/*********************************************************************/
/*                                                                   */
/*   Thresholding routine.                                           */
/*                                                                   */
/*   Usage : iathrl(lim, edge, pct, max, min);                       */
/*           lim  - (l_array) image buffer (input/output)            */
/*           edge - (int) image edge size (input)                    */
/*           pct  - (float) percentage from top value (input)        */
/*           max  - (long *) new maximum value (output)              */
/*           min  - (long *) new minimum value (output)              */
/*                                                                   */
/*********************************************************************/

char Imagemsg[5][20];

iathrl(liml, edge, pct, max, min)
char    *liml; /* image buffer (input/output)                  */
int    edge;   /* edge size of image (input)                   */
long   *max,   /* maximum value of threshold (output)          */
       *min;   /* minimum value of threshold (output)          */
float pct;     /* percentage of image for threshold (input)    */
{
  long i;      /* counter                          */
  long max1,   /* temp max storage                 */
       min1,   /* temp min storage                 */
       max2;   /* temp max storage                 */
 char  flg;    /* first time flag                  */
 register long far
       *ptr;   /* buffer pointer                   */

       /* set error messages */

  strcpy(Imagemsg[0], "Illegal i/o image buffer");
  strcpy(Imagemsg[1], "Illegal value for edge size");
  strcpy(Imagemsg[2], "Illegal value for percentage");
  strcpy(Imagemsg[3], "Illegal address for max value");
  strcpy(Imagemsg[4], "Illegal address for min value");

       /* check parameters */

  if (!liml)
    Imageerr(fname, 0, liml, lx);
  if (edge < 2)
    Imageerr(fname, 1, edge, d);
  if (pct < 0.0 || pct > 1.0)
    Imageerr(fname, 2, pct, f);
  if (!max)
    Imageerr(fname, 3, max, lx);
  if (!min)
    Imageerr(fname, 4, min, lx);

  flg = 1;
  max1 = min1 = 0;

       /* find min, max values of image */

  for (i=0; i<edge * edge; ++i,++ptr)
   {
    ptr = (long far *)aele(liml, i%edge, i/edge);
    if (*ptr > max1 || flg)
      max1 = *ptr;
    if (*ptr < min1 || flg)
      min1 = *ptr;
    if (flg)
      flg = 0;
   }

       /* determine percentage of actual values */

  max2 = (max1 - min1) * pct + min1;
  for (i=0; i<edge * edge; ++i,++ptr)
   {
    ptr = (long far *)aele(liml, i%edge, i/edge);
    if (*ptr <= max2 && *ptr > *max)
      *max = *ptr;
   }
  *min = min1;
}
```

IATHRL DL REPRESENTATION

```
Imageerr(char *, int, int);
strcpy(char *, char *);
printf(char *, int);
aele(char *, int, int);

char Imagemsg[5][20];
char fname[20];

function
iathrl(liml, edge, pct, max, min)
char    *liml;
int     edge;
long    *max,
        *min;
float pct;
{
  long i;
  long max1,
       min1,
       max2;
 char  flg;
 register long far
       *ptr;


  strcpy(Imagemsg[0], "Illegal i/o image buffer");
  strcpy(Imagemsg[1], "Illegal value for edge size");
  strcpy(Imagemsg[2], "Illegal value for percentage");
  strcpy(Imagemsg[3], "Illegal address for max value");
  strcpy(Imagemsg[4], "Illegal address for min value");



  if (!liml)
    Imageerr(fname, 0, edge);
  if (edge < 2)
    Imageerr(fname, 1, edge);
  if (pct < 0.0 || pct > 1.0)
    Imageerr(fname, 2, edge);
  if (!max)
    Imageerr(fname, 3, edge);
  if (!min)
    Imageerr(fname, 4, edge);

  flg = 1;
  max1 = min1 = 0;

  for (i=0; i<edge * edge; ++i,++ptr)
   {
    ptr = (long far *)aele(liml, i%edge, i/edge);
    if (*ptr > max1 || flg)
      max1 = *ptr;
    if (*ptr < min1 || flg)
      min1 = *ptr;
    if (flg)
      flg = 0;
   }

  max2 = (max1 - min1) * pct + min1;
  for (i=0; i<edge * edge; ++i,++ptr)
   {
    ptr = (long far *)aele(liml, i%edge, i/edge);
    if (*ptr <= max2 && *ptr > *max)
      *max = *ptr;
   }
  *min = min1;
}
```

IATHRL SYMBOL TABLE REPRESENTATION

```
Parsing input
   2:   3:   4:   5:   6:   7:   8:   9:  10:  11:  12:  13:  14:  15:  16:
  17:  18:  19:  20:  21:  22:  23:  24:  25:  26:  27:  28:  29:  30:  31:  32:
  33:  34:  35:  36:  37:  38:  39:  40:  41:  42:  43:  44:  45:  46:  47:  48:
  49:  50:  51:  52:  53:  54:  55:  56:  57:  58:  59:  60:  61:  62:  63:  64:
  65:  66:  67:  68:  69:  70:
Dumping all internal table information

Block Index
  Id# Level Name
  --- ----- ---------------
    0     0  anchor
    1     1    external
    2     2      Imageerr
    3     2      strcpy
    4     2      printf
    5     2      aele
    6     2      iathrl
    7     3        ?11
    8     3        ?13


Block # 0  anchor           (Level:  0)
  bl_type:   BT_BASE
  bl_return: <no return type>
  bl_parent: <no parent>
  bl_formal: <none>

TBL_TYPE   (used 18/25 entries) of OT_TYPE
Note: '!' after Size - full access requires far pointer.
 Id#  Name              Class      Size        References
----- ----------------- ---------- ----        ----------
 0/ 0  complex           TY_COMPLEX    8
 0/ 1  default           TY_ALIAS      2
       of TBL_TYPE  0/11
 0/ 2  default const     TY_CONST      2        Value: 0
 0/ 3  double            TY_DBL        8
 0/ 4  double complex    TY_DCMPLEX   16
 0/ 5  enumerator        TY_ENMRTR     4
 0/ 6  float             TY_FLT        4
 0/ 7  prototype far     TY_FAR        4
 0/ 8  prototype near    TY_NEAR       4
 0/ 9  signed char       TY_CHAR       1
 0/10  signed long       TY_LONG       4
 0/11  signed short      TY_SHORT      2
 0/12  unsigned char     TY_CHAR       1
 0/13  unsigned long     TY_LONG       4
 0/14  unsigned short    TY_SHORT      2
 0/15  void              TY_VOID       0
 0/16 int                TY_ALIAS      2
       of TBL_TYPE  0/ 1
 0/17 unsigned           TY_ALIAS      2
```

```
        of TBL_TYPE  0/14

TBL_BLOCK  (used 1/1 entries) of OT_BLOCK
 Id # Blk # Name
----- ----- ---------------
 0/ 0     1  external


Block # 1  external          (Level:  1)
  bl_type:   BT_BLOCK
  bl_return: <no return type>
  bl_parent: anchor
  bl_formal: <none>

TBL_TYPE   (used 6/9 entries) of OT_TYPE
Note: '!' after Size - full access requires far pointer.
 Id#  Name              Class       Size        References
----- ----------------- ---------- ----        ----------------
 1/ 0  () returning     TY_FUNC       2        TBL_TYPE  0/ 1 ()
 1/ 1  [0 ... 19]       TY_ARR       20     of TBL_TYPE  0/ 9 [0 ... 19]
 1/ 2  [0 ... 19]       TY_ARR       20     of TBL_TYPE  0/ 9 [0 ... 19]
 1/ 3  [0 ... 4]        TY_ARR      100     of TBL_TYPE  1/ 2 [0 ... 4];[0 ... 19]
 1/ 4  near pointer     TY_NEAR       4     to TBL_TYPE  0/ 9 near/volatile
 1/ 5  near pointer     TY_NEAR       4     to TBL_TYPE  0/10 near/volatile

TBL_CONST  (used 4/9 entries) of OT_TYPE
Note: '!' after Size - full access requires far pointer.
 Id#  Name              Class       Size        References
----- ----------------- ---------- ----        ----------------
 1/ 0  upper bound      TY_CONST      4        Value: 19
 1/ 1  upper bound      TY_CONST      4        Value: 19
 1/ 2  long constant    TY_CONST      4        Value: 0
 1/ 3  upper bound      TY_CONST      4        Value: 4

TBL_VAR    (used 7/9 entries) of OT_SYMBOL
 Id#  Name              Usage      Type      Attributes/References
----- ----------------- --------  -----     ---------------------
 1/ 0 Imageerr          US_DECL    1/ 0
 1/ 1 Imagemsg          US_DECL    1/ 3
 1/ 2 aele              US_DECL    1/ 0
 1/ 3 fname             US_DECL    1/ 1
 1/ 4 iathrl            US_DECL    1/ 0
 1/ 5 printf            US_DECL    1/ 0
 1/ 6 strcpy            US_DECL    1/ 0

TBL_BLOCK  (used 5/9 entries) of OT_BLOCK
 Id # Blk # Name
----- ----- ---------------
 1/ 0     2  Imageerr
 1/ 1     3  strcpy
 1/ 2     4  printf
 1/ 3     5  aele
```

```
 1/ 4    6  iathrl


Block # 2  Imageerr          (Level:  2)
  bl_type:   BT_PROTO
  bl_return:  default
  bl_parent: external
  bl_formal:  TBL_VAR  2/ 0 TBL_VAR  2/ 1 TBL_VAR  2/ 2

TBL_VAR    (used 3/9 entries) of OT_SYMBOL
 Id#  Name              Usage      Type      Attributes/References
----- ---------------- --------  -----     ----------------------
 2/ 0 ?0               US_FPARM   1/ 4
 2/ 1 ?1               US_FPARM   0/16
 2/ 2 ?2               US_FPARM   0/16


Block # 3  strcpy            (Level:  2)
  bl_type:   BT_PROTO
  bl_return:  default
  bl_parent: external
  bl_formal:  TBL_VAR  3/ 0 TBL_VAR  3/ 1

TBL_VAR    (used 2/9 entries) of OT_SYMBOL
 Id#  Name              Usage      Type      Attributes/References
----- ---------------- --------  -----     ----------------------
 3/ 0 ?3               US_FPARM   1/ 4
 3/ 1 ?4               US_FPARM   1/ 4


Block # 4  printf            (Level:  2)
  bl_type:   BT_PROTO
  bl_return:  default
  bl_parent: external
  bl_formal:  TBL_VAR  4/ 0 TBL_VAR  4/ 1

TBL_VAR    (used 2/9 entries) of OT_SYMBOL
 Id#  Name              Usage      Type      Attributes/References
----- ---------------- --------  -----     ----------------------
 4/ 0 ?5               US_FPARM   1/ 4
 4/ 1 ?6               US_FPARM   0/16


Block # 5  aele              (Level:  2)
  bl_type:   BT_PROTO
  bl_return:  default
  bl_parent: external
  bl_formal:  TBL_VAR  5/ 0 TBL_VAR  5/ 1 TBL_VAR  5/ 2

TBL_VAR    (used 3/9 entries) of OT_SYMBOL
 Id#  Name              Usage      Type      Attributes/References
----- ---------------- --------  -----     ----------------------
```

```
 5/ 0 77              US_FPARM   1/ 4
 5/ 1 78              US_FPARM   0/16
 5/ 2 79              US_FPARM   0/16


Block # 6  iathrl          (Level:  2)
  bl_type:   BT_FUNC
  bl_return:  default
  bl_parent: external
  bl_formal:  TBL_VAR  6/ 3 TBL_VAR  6/ 0 TBL_VAR  6/ 9 TBL_VAR  6/ 4 TBL_VAR  6/ 7


TBL_TYPE   (used 1/1 entries) of OT_TYPE
Note: '!' after Size - full access requires far pointer.
 Id#  Name              Class      Size        References
----- ----------------- ---------- ----        ----------
 6/ 0   far pointer     TY_FAR        4     to TBL_TYPE  0/10 far/volatile

TBL_CONST  (used 12/17 entries) of OT_TYPE
Note: '!' after Size - full access requires far pointer.
 Id#  Name              Class      Size        References
----- ----------------- ---------- ----        ----------
 6/ 0  double constant  TY_CONST      8        Value:  1.000000e+00
 6/ 1  double constant  TY_CONST      8        Value:  0.000000e+00
 6/ 2  long constant    TY_CONST      4        Value: 1
 6/ 3  long constant    TY_CONST      4        Value: 2
 6/ 4  long constant    TY_CONST      4        Value: 3
 6/ 5  long constant    TY_CONST      4        Value: 4
 6/ 6  long constant    TY_CONST      4        Value: 0
 6/ 7  string constant  TY_CONST     28        Value: "Illegal value for percentage"
 6/ 8  string constant  TY_CONST     29        Value: "Illegal address for min value"
 6/ 9  string constant  TY_CONST     29        Value: "Illegal address for max value"
 6/10  string constant  TY_CONST     27        Value: "Illegal value for edge size"
 6/11  string constant  TY_CONST     24        Value: "Illegal i/o image buffer"

TBL_VAR    (used 11/17 entries) of OT_SYMBOL
 Id#  Name              Usage      Type     Attributes/References
----- ----------------- --------   -----    ---------------------
 6/ 0 edge              US_FPARM   0/16
 6/ 1 flg               US_DECL    0/ 9
 6/ 2 i                 US_DECL    0/10
 6/ 3 lim1              US_FPARM   1/ 4
 6/ 4 max               US_FPARM   1/ 5
 6/ 5 max1              US_DECL    0/10
 6/ 6 max2              US_DECL    0/10
 6/ 7 min               US_FPARM   1/ 5
 6/ 8 min1              US_DECL    0/10
 6/ 9 pct               US_FPARM   0/ 6
 6/10 ptr               US_DECL    6/ 0     Attributes:
                                             register

TBL_CODE   (used 11/17 entries) of OT_TYPE
```

```
 Id#  Quadruple Sequence
----- --------------------
 6/ 0     001|00000014 TBL_CODE  6/ 1 TBL_CODE  6/ 2 TBL_CODE  6/ 3 TBL_CODE  6/ 4
      TBL_CODE  6/ 5 TBL_QUAD  6/25 TBL_QUAD  6/31 TBL_QUAD  6/39
      TBL_QUAD  6/45 TBL_QUAD  6/51 TBL_QUAD  6/52 TBL_QUAD  6/54
      TBL_QUAD  6/56 TBL_QUAD  6/67 TBL_QUAD  6/69 TBL_QUAD  6/78

 6/ 1     002|00000000 TBL_QUAD  6/ 1 TBL_QUAD  6/ 2 TBL_QUAD  6/ 3

 6/ 2     002|00000001 TBL_QUAD  6/ 5 TBL_QUAD  6/ 6 TBL_QUAD  6/ 7

 6/ 3     002|00000002 TBL_QUAD  6/ 9 TBL_QUAD  6/10 TBL_QUAD  6/11

 6/ 4     002|00000003 TBL_QUAD  6/13 TBL_QUAD  6/14 TBL_QUAD  6/15

 6/ 5     002|00000004 TBL_QUAD  6/17 TBL_QUAD  6/18 TBL_QUAD  6/19

 6/ 6     003|00000005 TBL_QUAD  6/21 TBL_QUAD  6/22 TBL_QUAD  6/23 TBL_QUAD  6/24

 6/ 7     003|00000006 TBL_QUAD  6/27 TBL_QUAD  6/28 TBL_QUAD  6/29 TBL_QUAD  6/30

 6/ 8     003|00000007 TBL_QUAD  6/35 TBL_QUAD  6/36 TBL_QUAD  6/37 TBL_QUAD  6/38

 6/ 9     003|00000008 TBL_QUAD  6/41 TBL_QUAD  6/42 TBL_QUAD  6/43 TBL_QUAD  6/44

 6/10     003|00000009 TBL_QUAD  6/47 TBL_QUAD  6/48 TBL_QUAD  6/49 TBL_QUAD  6/50

TBL_QUAD   (used 79/81 entries) of OT_QUAD
 Id #  Operation       Left            Right           Third          Result
----- ---------- --------------- --------------- --------------- ---------------
 6/ 0         [] TBL_VAR    1/ 1 TBL_CONST  6/ 6  < none >       TBL_TYPE   1/ 2
 6/ 1       push TBL_TYPE   1/ 4 TBL_QUAD   6/ 0  < none >       TBL_TYPE   1/ 4
      warning, automatic cast to result type!
 6/ 2       push TBL_TYPE   1/ 4 TBL_CONST  6/11  < none >       TBL_TYPE   1/ 4
 6/ 3       call TBL_VAR    1/ 6  < none >        < none >       TBL_TYPE   0/ 1
 6/ 4         [] TBL_VAR    1/ 1 TBL_CONST  6/ 2  < none >       TBL_TYPE   1/ 2
 6/ 5       push TBL_TYPE   1/ 4 TBL_QUAD   6/ 4  < none >       TBL_TYPE   1/ 4
      warning, automatic cast to result type!
 6/ 6       push TBL_TYPE   1/ 4 TBL_CONST  6/10  < none >       TBL_TYPE   1/ 4
 6/ 7       call TBL_VAR    1/ 6  < none >        < none >       TBL_TYPE   0/ 1
 6/ 8         [] TBL_VAR    1/ 1 TBL_CONST  6/ 3  < none >       TBL_TYPE   1/ 2
 6/ 9       push TBL_TYPE   1/ 4 TBL_QUAD   6/ 8  < none >       TBL_TYPE   1/ 4
      warning, automatic cast to result type!
 6/10       push TBL_TYPE   1/ 4 TBL_CONST  6/ 7  < none >       TBL_TYPE   1/ 4
 6/11       call TBL_VAR    1/ 6  < none >        < none >       TBL_TYPE   0/ 1
 6/12         [] TBL_VAR    1/ 1 TBL_CONST  6/ 4  < none >       TBL_TYPE   1/ 2
 6/13       push TBL_TYPE   1/ 4 TBL_QUAD   6/12  < none >       TBL_TYPE   1/ 4
      warning, automatic cast to result type!
 6/14       push TBL_TYPE   1/ 4 TBL_CONST  6/ 9  < none >       TBL_TYPE   1/ 4
 6/15       call TBL_VAR    1/ 6  < none >        < none >       TBL_TYPE   0/ 1
 6/16         [] TBL_VAR    1/ 1 TBL_CONST  6/ 5  < none >       TBL_TYPE   1/ 2
```

```
6/17        push TBL_TYPE   1/ 4 TBL_QUAD   6/16  < none >        TBL_TYPE   1/ 4
    warning, automatic cast to result type!
6/18        push TBL_TYPE   1/ 4 TBL_CONST  6/ 8  < none >        TBL_TYPE   1/ 4
6/19        call TBL_VAR    1/ 6  < none >        < none >        TBL_TYPE   0/ 1
6/20           ! TBL_VAR    6/ 3  < none >        < none >        TBL_TYPE   0/ 1
6/21        push TBL_TYPE   1/ 4 TBL_VAR    1/ 3  < none >        TBL_TYPE   1/ 4
    warning, automatic cast to result type!
6/22        push TBL_TYPE   0/16 TBL_CONST  6/ 6  < none >        TBL_TYPE   0/11
6/23        push TBL_TYPE   0/16 TBL_VAR    6/ 0  < none >        TBL_TYPE   0/11
6/24        call TBL_VAR    1/ 0  < none >        < none >        TBL_TYPE   0/ 1
6/25          if TBL_QUAD   6/20 TBL_CODE   6/ 6  < none >         < none >
6/26           < TBL_VAR    6/ 0 TBL_CONST  6/ 3  < none >        TBL_TYPE   0/ 1
6/27        push TBL_TYPE   1/ 4 TBL_VAR    1/ 3  < none >        TBL_TYPE   1/ 4
    warning, automatic cast to result type!
6/28        push TBL_TYPE   0/16 TBL_CONST  6/ 2  < none >        TBL_TYPE   0/11
6/29        push TBL_TYPE   0/16 TBL_VAR    6/ 0  < none >        TBL_TYPE   0/11
6/30        call TBL_VAR    1/ 0  < none >        < none >        TBL_TYPE   0/ 1
6/31          if TBL_QUAD   6/26 TBL_CODE   6/ 7  < none >         < none >
6/32           < TBL_VAR    6/ 9 TBL_CONST  6/ 1  < none >        TBL_TYPE   0/ 1
6/33           > TBL_VAR    6/ 9 TBL_CONST  6/ 0  < none >        TBL_TYPE   0/ 1
6/34          || TBL_QUAD   6/32 TBL_QUAD   6/33  < none >        TBL_TYPE   0/ 1
6/35        push TBL_TYPE   1/ 4 TBL_VAR    1/ 3  < none >        TBL_TYPE   1/ 4
    warning, automatic cast to result type!
6/36        push TBL_TYPE   0/16 TBL_CONST  6/ 3  < none >        TBL_TYPE   0/11
6/37        push TBL_TYPE   0/16 TBL_VAR    6/ 0  < none >        TBL_TYPE   0/11
6/38        call TBL_VAR    1/ 0  < none >        < none >        TBL_TYPE   0/ 1
6/39          if TBL_QUAD   6/34 TBL_CODE   6/ 8  < none >         < none >
6/40           ! TBL_VAR    6/ 4  < none >        < none >        TBL_TYPE   0/ 1
6/41        push TBL_TYPE   1/ 4 TBL_VAR    1/ 3  < none >        TBL_TYPE   1/ 4
    warning, automatic cast to result type!
6/42        push TBL_TYPE   0/16 TBL_CONST  6/ 4  < none >        TBL_TYPE   0/11
6/43        push TBL_TYPE   0/16 TBL_VAR    6/ 0  < none >        TBL_TYPE   0/11
6/44        call TBL_VAR    1/ 0  < none >        < none >        TBL_TYPE   0/ 1
6/45          if TBL_QUAD   6/40 TBL_CODE   6/ 9  < none >         < none >
6/46           ! TBL_VAR    6/ 7  < none >        < none >        TBL_TYPE   0/ 1
6/47        push TBL_TYPE   1/ 4 TBL_VAR    1/ 3  < none >        TBL_TYPE   1/ 4
    warning, automatic cast to result type!
6/48        push TBL_TYPE   0/16 TBL_CONST  6/ 5  < none >        TBL_TYPE   0/11
6/49        push TBL_TYPE   0/16 TBL_VAR    6/ 0  < none >        TBL_TYPE   0/11
6/50        call TBL_VAR    1/ 0  < none >        < none >        TBL_TYPE   0/ 1
6/51          if TBL_QUAD   6/46 TBL_CODE   6/10  < none >         < none >
6/52           = TBL_VAR    6/ 1 TBL_CONST  6/ 2  < none >        TBL_TYPE   0/ 9
    warning, automatic cast to result type!
6/53           = TBL_VAR    6/ 8 TBL_CONST  6/ 6  < none >        TBL_TYPE   0/10
6/54           = TBL_VAR    6/ 5 TBL_QUAD   6/53  < none >        TBL_TYPE   0/10
6/55           = TBL_VAR    6/ 2 TBL_CONST  6/ 6  < none >        TBL_TYPE   0/10
6/56         for TBL_QUAD   6/55 TBL_QUAD   6/63  < none >         < none >
6/57           * TBL_VAR    6/ 0 TBL_VAR    6/ 0  < none >        TBL_TYPE   0/11
6/58           < TBL_VAR    6/ 2 TBL_QUAD   6/57  < none >        TBL_TYPE   0/ 1
6/59         ++x TBL_VAR    6/ 2  < none >        < none >        TBL_TYPE   0/10
6/60         ++x TBL_VAR    6/10  < none >        < none >        TBL_TYPE   6/ 0
6/61           , TBL_QUAD   6/59 TBL_QUAD   6/60  < none >         < none >
```

```
6/62        call TBL_BLOCK  6/ 0  < none >        < none >        < none >
6/63       while TBL_QUAD   6/58 TBL_QUAD   6/61 TBL_QUAD   6/62  < none >
6/64           - TBL_VAR    6/ 5 TBL_VAR    6/ 8  < none >       TBL_TYPE   0/10
6/65           * TBL_QUAD   6/64 TBL_VAR    6/ 9  < none >       TBL_TYPE   0/ 6
     warning, automatic cast to result type!
6/66           + TBL_QUAD   6/65 TBL_VAR    6/ 8  < none >       TBL_TYPE   0/ 6
     warning, automatic cast to result type!
6/67           = TBL_VAR    6/ 6 TBL_QUAD   6/66  < none >       TBL_TYPE   0/10
     warning, automatic cast to result type!
6/68           = TBL_VAR    6/ 2 TBL_CONST  6/ 6  < none >       TBL_TYPE   0/10
6/69         for TBL_QUAD   6/68 TBL_QUAD   6/76  < none >        < none >
6/70           * TBL_VAR    6/ 0 TBL_VAR    6/ 0  < none >       TBL_TYPE   0/11
6/71           < TBL_VAR    6/ 2 TBL_QUAD   6/70  < none >       TBL_TYPE   0/ 1
6/72         ++x TBL_VAR    6/ 2  < none >        < none >       TBL_TYPE   0/10
6/73         ++x TBL_VAR    6/10  < none >        < none >       TBL_TYPE   6/ 0
6/74           , TBL_QUAD   6/72 TBL_QUAD   6/73  < none >        < none >
6/75        call TBL_BLOCK  6/ 1  < none >        < none >        < none >
6/76       while TBL_QUAD   6/71 TBL_QUAD   6/74 TBL_QUAD   6/75  < none >
6/77         *() TBL_VAR    6/ 7  < none >        < none >       TBL_TYPE   0/10
6/78           = TBL_QUAD   6/77 TBL_VAR    6/ 8  < none >       TBL_TYPE   0/10

TBL_BLOCK  (used 2/9 entries) of OT_BLOCK
 Id # Blk # Name
----- ----- ---------------
 6/ 0    7  ?11
 6/ 1    8  ?13


Block # 7  ?11              (Level:  3)
  bl_type:   BT_SUBR
  bl_return: <no return type>
  bl_parent: iathrl
  bl_formal: <none>

TBL_CONST  (used 1/1 entries) of OT_TYPE
Note: '!' after Size - full access requires far pointer.
 Id#  Name             Class      Size        References
----- ---------------- ---------- ----        ----------
 7/ 0  long constant   TY_CONST      4        Value: 0

TBL_CODE   (used 2/9 entries) of OT_TYPE
 Id#  Quadruple Sequence
----- -------------------
 7/ 0     002|00000011 TBL_QUAD  7/ 7 TBL_QUAD  7/13 TBL_QUAD  7/19 TBL_QUAD  7/21

 7/ 1     003|00000010 TBL_QUAD  7/ 2 TBL_QUAD  7/ 3 TBL_QUAD  7/ 4 TBL_QUAD  7/ 5


TBL_QUAD   (used 22/25 entries) of OT_QUAD
 Id #  Operation      Left           Right          Third          Result
----- ---------- -------------- -------------- -------------- --------------
 7/ 0          % TBL_VAR    6/ 2 TBL_VAR    6/ 0  < none >       TBL_TYPE   0/10
```

```
      warning, automatic cast to result type!
 7/ 1          / TBL_VAR    6/ 2 TBL_VAR    6/ 0  < none >       TBL_TYPE   0/10
      warning, automatic cast to result type!
 7/ 2       push TBL_TYPE   1/ 4 TBL_VAR    6/ 3  < none >       TBL_TYPE   1/ 4
 7/ 3       push TBL_TYPE   0/16 TBL_QUAD   7/ 0  < none >       TBL_TYPE   0/11
      warning, automatic cast to result type!
 7/ 4       push TBL_TYPE   0/16 TBL_QUAD   7/ 1  < none >       TBL_TYPE   0/11
      warning, automatic cast to result type!
 7/ 5       call TBL_VAR    1/ 2  < none >        < none >       TBL_TYPE   0/ 1
 7/ 6     (cast) TBL_TYPE   6/ 0 TBL_CODE   7/ 1  < none >       TBL_TYPE   6/ 0
 7/ 7          = TBL_VAR    6/10 TBL_QUAD   7/ 6  < none >       TBL_TYPE   1/ 5
 7/ 8        *() TBL_VAR    6/10  < none >        < none >       TBL_TYPE   0/10
 7/ 9          > TBL_QUAD   7/ 8 TBL_VAR    6/ 5  < none >       TBL_TYPE   0/ 1
 7/10         || TBL_QUAD   7/ 9 TBL_VAR    6/ 1  < none >       TBL_TYPE   0/ 1
 7/11        *() TBL_VAR    6/10  < none >        < none >       TBL_TYPE   0/10
 7/12          = TBL_VAR    6/ 5 TBL_QUAD   7/11  < none >       TBL_TYPE   0/10
 7/13         if TBL_QUAD   7/10 TBL_QUAD   7/12  < none >        < none >
 7/14        *() TBL_VAR    6/10  < none >        < none >       TBL_TYPE   0/10
 7/15          < TBL_QUAD   7/14 TBL_VAR    6/ 8  < none >       TBL_TYPE   0/ 1
 7/16         || TBL_QUAD   7/15 TBL_VAR    6/ 1  < none >       TBL_TYPE   0/ 1
 7/17        *() TBL_VAR    6/10  < none >        < none >       TBL_TYPE   0/10
 7/18          = TBL_VAR    6/ 8 TBL_QUAD   7/17  < none >       TBL_TYPE   0/10
 7/19         if TBL_QUAD   7/16 TBL_QUAD   7/18  < none >        < none >
 7/20          = TBL_VAR    6/ 1 TBL_CONST  7/ 0  < none >       TBL_TYPE   0/ 9
      warning, automatic cast to result type!
 7/21         if TBL_VAR    6/ 1 TBL_QUAD   7/20  < none >        < none >


Block # 8  ?13              (Level:  3)
  bl_type:   BT_SUBR
  bl_return: <no return type>
  bl_parent: isthrl
  bl_formal: <none>

TBL_CODE   (used 2/9 entries) of OT_TYPE
 Id#  Quadruple Sequence
----- ------------------
 8/ 0     002¦00000013 TBL_QUAD  8/ 7 TBL_QUAD  8/17

 8/ 1     003¦00000012 TBL_QUAD  8/ 2 TBL_QUAD  8/ 3 TBL_QUAD  8/ 4 TBL_QUAD  8/ 5


TBL_QUAD   (used 18/25 entries) of OT_QUAD
 Id #  Operation       Left            Right           Third          Result
----- --------- --------------- --------------- --------------- ---------------
 8/ 0          % TBL_VAR    6/ 2 TBL_VAR    6/ 0  < none >       TBL_TYPE   0/10
      warning, automatic cast to result type!
 8/ 1          / TBL_VAR    6/ 2 TBL_VAR    6/ 0  < none >       TBL_TYPE   0/10
      warning, automatic cast to result type!
 8/ 2       push TBL_TYPE   1/ 4 TBL_VAR    6/ 3  < none >       TBL_TYPE   1/ 4
 8/ 3       push TBL_TYPE   0/16 TBL_QUAD   8/ 0  < none >       TBL_TYPE   0/11
      warning, automatic cast to result type!
```

```
8/ 4        push TBL_TYPE   0/16 TBL_QUAD    8/ 1  < none >        TBL_TYPE    0/11
    warning, automatic cast to result type!
8/ 5        call TBL_VAR    1/ 2  < none >                  < none >
8/ 6     (cast) TBL_TYPE    6/ 0 TBL_CODE    8/ 1  < none >        TBL_TYPE    0/ 1
8/ 7           = TBL_VAR    6/10 TBL_QUAD    8/ 6  < none >        TBL_TYPE    6/ 0
8/ 8        *() TBL_VAR     6/10  < none >                  < none >        TBL_TYPE    1/ 5
8/ 9         <= TBL_QUAD    8/ 8 TBL_VAR     6/ 6  < none >        TBL_TYPE    0/10
8/10        *() TBL_VAR     6/10  < none >                  < none >        TBL_TYPE    0/ 1
8/11        *() TBL_VAR     6/ 4  < none >                  < none >        TBL_TYPE    0/10
8/12         >  TBL_QUAD    8/10 TBL_QUAD    8/11  < none >        TBL_TYPE    0/10
8/13         && TBL_QUAD    8/ 9 TBL_QUAD    8/12  < none >        TBL_TYPE    0/ 1
8/14        *() TBL_VAR     6/ 4  < none >                  < none >        TBL_TYPE    0/ 1
8/15        *() TBL_VAR     6/10  < none >                  < none >        TBL_TYPE    0/10
8/16           = TBL_QUAD   8/14 TBL_QUAD    8/15  < none >        TBL_TYPE    0/10
8/17          if TBL_QUAD   8/13 TBL_QUAD    8/16  < none >        TBL_TYPE    0/10
                                                                    < none >
```

```
subroutine
  iathrl,
  return_type  default,
  called_by
    ' ',
  calls
    strcpy,
    Imageerr,
    aele,
  params
    ' signed short edge',
    ' signed char  *liml',
    ' signed long  *max',
    ' signed long  *min',
    ' float pct',
  ext_vars
    ' ',
  loc_vars
    ' signed char flg',
    ' signed long i',
    ' signed long max1',
    ' signed long max2',
    ' signed long min1',
    ' signed long  *ptr',
  metrics
    ' ',
{
  strcpy(Imagemsg[0], "Illegal i/o image buffer");
  strcpy(Imagemsg[1], "Illegal value for edge size");
  strcpy(Imagemsg[2], "Illegal value for percentage");
  strcpy(Imagemsg[3], "Illegal address for max value");
  strcpy(Imagemsg[4], "Illegal address for min value");
  page_break P1,
}

page P1,
{
  if (!liml)
    Imageerr(fname, 0, edge);
  else
    ( )
  if (edge < 1)
    Imageerr(fname, 1, edge);
  else
    ( )
  if (pct <  0.000000e+00 || pct >  1.000000e+00)
    Imageerr(fname, 2, edge);
  else
    ( )
  if (!max)
    Imageerr(fname, 3, edge);
  else
    ( )
  if (!min)
    Imageerr(fname, 4, edge);
  else
    ( )
  flg = 1;
  max1 = min1 = 0;
  i = 0;
  while (i < edge * edge)
    {
      ptr = ( signed long  *)aele(liml, i % edge, i / edge);
      if (*ptr > max1 || flg)
        max1 = *ptr;
      else
        ( )
      page_break P2,
    }
  max2 = (max1 - min1) * pct + min1;
  i = 0;
  page_break P3,
}

page P2,
{
  if (*ptr < min1 || flg)
    min1 = *ptr;
  else
    ( )
  if (flg)
    flg = 0;
  else
    ( )
  ++i , ++ptr;
}

page P3,
{
  while (i < edge * edge)
    {
      ptr = ( signed long  *)aele(liml, i % edge, i / edge);
      if (*ptr <= max2 && *ptr > *max)
        *max = *ptr;
      else
        ( )
      ++i , ++ptr;
    }
  *min = min1;
}
```

THE RESULTING NASSI-SHNEIDERMANN CHART ❶

iathrl	*Return Type* default
Called By	*Calls* strcpy Imageerr aele
Parameters signed short edge signed char *lim1 signed long *max signed long *min float pct	*Metrics*
External Variables Affected	
Local Variables signed char flg signed long i signed long max1 signed long max2 signed long min1 signed long *ptr	

strcpy(Imagemsg[0], "Illegal i/o image buffer")

strcpy(Imagemsg[1], "Illegal value for edge size")

strcpy(Imagemsg[2], "Illegal value for percentage")

strcpy(Imagemsg[3], "Illegal address for max value")

strcpy(Imagemsg[4], "Illegal address for min value")

❷

Continued from page 1 ❷

(!lim1)
Yes: Imageerr(fname, 0, edge)
No:

(edge < 2)
Yes: Imageerr(fname, 1, edge)
No:

(pct < 0.000000e+00 || pct > 1.000000e+00)
Yes: Imageerr(fname, 2, edge)
No:

(!max)
Yes: Imageerr(fname, 3, edge)
No:

(!min)
Yes: Imageerr(fname, 4, edge)
No:

flg = 1

max1 = min1 = 0

i = 0

While (i < edge * edge)

ptr = (signed long *)aele(lim1, i % edge, i / edge)

(*ptr > max1 || flg)
Yes: max1 = *ptr
No:

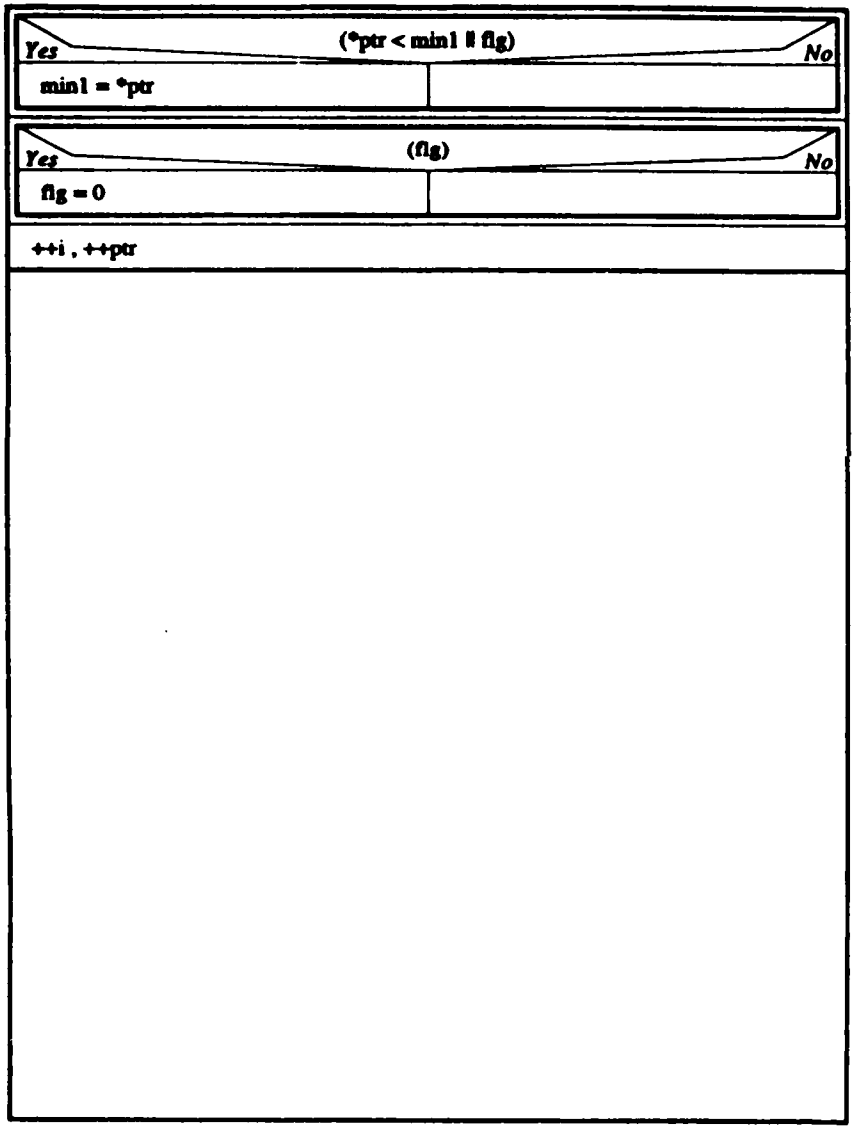
❸

max2 = (max1 - min1) * pct + min1

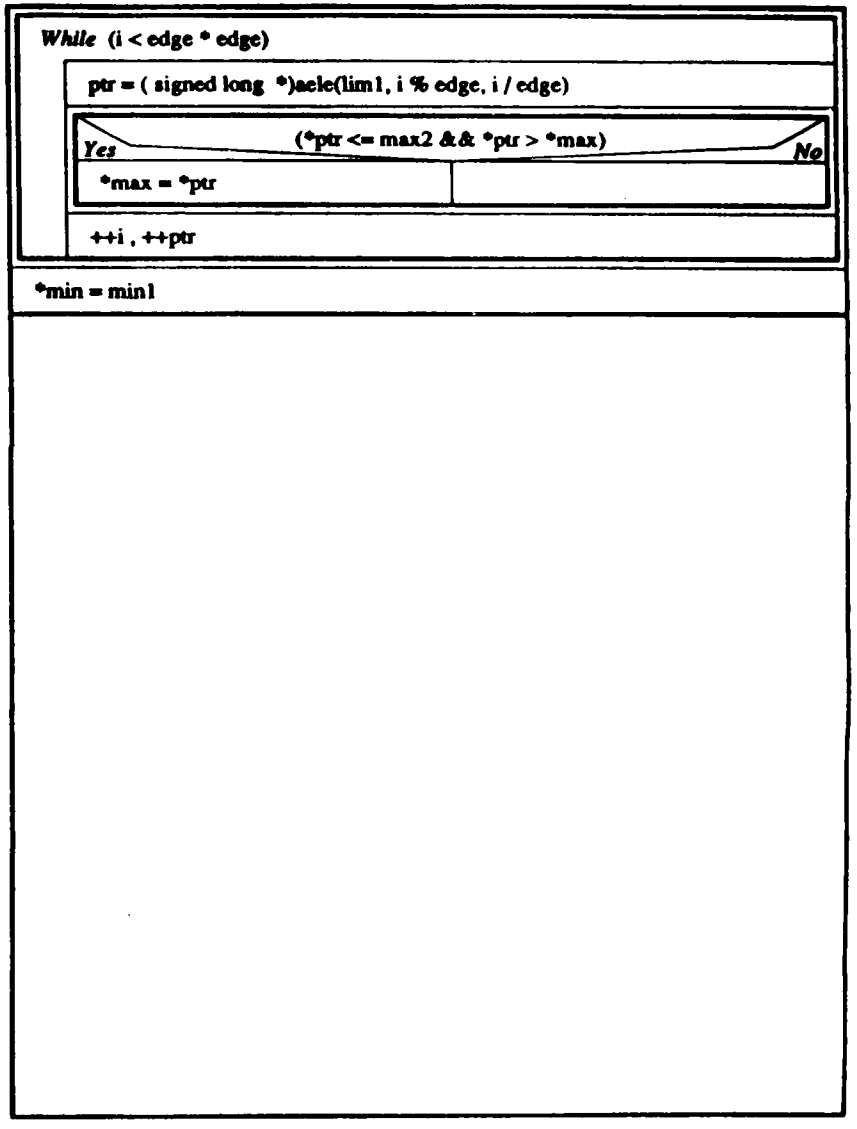
i = 0

❹

Continued from page 2 ❸



Continued from page 2 ❸



APPENDIX C

MAIN EXAMPLE

THE C-CODED ROUTINE

```
/*
main();
sub1(int, int);
sub2(int, int);
print2(char *, char *, char *);

printf(char *);
calloc(int, int);
*/
char *strcat();
char *strcpy();

main()
{
  printf("Running main program\n");
  sub1(5, 3);
  sub2(10, 4);
  printf("Main program finished\n");
}

sub1(p1, p2)
int p1;
int p2;
{
  short i, j, k, x, y;
  char str[100];

  printf("Executing sub1\n");
  for (i=0; i<1; i++)
    {
      for (j=0; j<1; j++)
       {
          for (k=0; k<1; k++)
             printf("loop line 1\n");
         printf("loop line 2\n");
       }
      printf("loop line 3\n");
     y = 3;
     x = !y;
     x = ~y;
     x = str[0];
     x = j;
    )
   x += y;
   x -= y;
   x *= y;
   x /= y;
   x &= y;
   x = !y;
   x = ~y;
   x = str[0];
   x = j;
   x += y;
   x -= y;
   x *= y;
   x /= y;
   x &= y;
   x = 2;
  switch (x)
    {
      case 1:
        y = 1;
       printf("first case stmt\n");
      case 2:
       printf("second case stmt\n");
        y = 2;
        break;
      case 3:
        x = 3;
       strcpy(str, "Case 3 of switch");
       break;
      default:
       strcpy(str, "default case");
        y = x + 1;
       printf("last case stmt\n");
    }
  return (j * (k + i));
}

sub2(param1, param2)
int param1;
int param2;
{
  long i, j;
  char *str1, *str2, *str3;
  int k;

  printf("Executing sub2\n");
  for (i=0; i<5; i++)
    do
      {
       j = 30;
       while (j <= 130)
         {
           for (k=0; k<100;)
             {
               printf("inner loop line\n");
               k = j * 2 + (i + 10);
               j += 50;
             }
           j += 10;
```

```
        }
        i++;
        printf("End of DO\n");
      }
    while (i < 10);
  str1 = (char *)calloc(20, 1);
  str2 = (char *)calloc(20, 1);
  str3 = (char *)calloc(20, 1);
  strcpy(str1, "first copy");
  str2[0] = '\0';
  print2(str1, strcat(str2, strcpy(str3, " cat1")), "prt1");
}

print2(str1, str2, str3)
char *str1;
char *str2;
char *str3;
{
  printf(str1);
  printf(str2);
  printf(str3);
  printf("\n");
}
```

MAIN DL REPRESENTATION

```
main();
sub1(int, int);
sub2(int, int);
print2(char *, char *, char *);

printf(char *);
calloc(int, int);

char *strcat(char *, char *);
char *strcpy(char *, char *);

function main()
{
  printf("Running main program\n");
  sub1(5, 3);
  sub2(10, 4);
  printf("Main program finished\n");
}


function sub1(p1, p2)
int p1;
int p2;
{
  short i, j, k, x, y;
  char str[100];

  printf("Executing sub1\n");
  for (i=0; i<1; i++)
    {
      for (j=0; j<1; j++)
       {
          for (k=0; k<1; k++)
             printf("loop line 1\n");
         printf("loop line 2\n");
       }
      printf("loop line 3\n");
     y = 3;
     x = !y;
     x = ~y;
     x = str[0];
     x = j;
    }
   x += y;
   x -= y;
   x *= y;
   x /= y;
   x &= y;
   x = !y;
   x = ~y;
   x = str[0];
   x = j;
   x += y;
   x -= y;
   x *= y;
   x /= y;
   x &= y;
   x = 2;
  switch (x)
    {
      case 1:
        y = 1;
       printf("first case stmt\n");
      case 2:
       printf("second case stmt\n");
        y = 2;
        break;
      case 3:
        x = 3;
       strcpy(str, "Case 3 of switch");
       break;
      default:
       strcpy(str, "default case");
        y = x + 1;
       printf("last case stmt\n");
    }
  return (j * (k + i));
}


function sub2(param1, param2)
int param1;
int param2;
{
  long i, j;
  char *str1, *str2, *str3;
  int k;

  printf("Executing sub2\n");
  for (i=0; i<5; i++)
    do
      {
       j = 30;
       while (j <= 130)
         {
           for (k=0; k<100;)
             {
               printf("inner loop line\n");
               k = j * 2 + (i + 10);
               j += 50;
             }
           j += 10;
```

```
        }
      i++;
      printf("End of DO\n");
     }
   while (i < 10);
  str1 = (char *)calloc(20, 1);
  str2 = (char *)calloc(20, 1);
  str3 = (char *)calloc(20, 1);
  strcpy(str1, "first copy");
  str2[0] = '\0';
  print2(str1, strcat(str2, strcpy(str3, " cat1")), "prt1");
}

function print2(str1, str2, str3)
char *str1;
char *str2;
char *str3;
{
  printf(str1);
  printf(str2);
  printf(str3);
  printf("\n");
}
```

MAIN SYMBOL TABLE REPRESENTATION

```
Parsing input
   2:   3:   4:   5:   6:   7:   8:   9:  10:  11:  12:  13:  14:  15:  16:
  17:  18:  19:  20:  21:  22:  23:  24:  25:  26:  27:  28:  29:  30:  31:  32:
  33:  34:  35:  36:  37:  38:  39:  40:  41:  42:  43:  44:  45:  46:  47:  48:
  49:  50:  51:  52:  53:  54:  55:  56:  57:  58:  59:  60:  61:  62:  63:  64:
  65:  66:  67:  68:  69:  70:  71:  72:  73:  74:  75:  76:  77:  78:  79:  80:
  81:  82:  83:  84:  85:  86:  87:  88:  89:  90:  91:  92:  93:  94:  95:  96:
  97:  98:  99: 100: 101: 102: 103: 104: 105: 106: 107: 108: 109: 110: 111: 112:
 113: 114: 115: 116: 117: 118: 119: 120: 121: 122: 123: 124: 125: 126: 127: 128:

Dumping all internal table information

Block Index
  Id# Level Name
  --- ----- ---------------
    0     0  anchor
    1     1    external
    2     2      main
    3     2      sub1
   12     3        ?15
   13     4          ?14
   14     3        ?16
    4     2      sub2
   16     3        ?19
   17     4          ?18
   18     5            ?17
    5     2      print2
    6     2      printf
    7     2      calloc
    8     2      strcat
    9     2      strcpy


 Block # 0  anchor            (Level:  0)
   bl_type:   BT_BASE
   bl_return: <no return type>
   bl_parent: <no parent>
   bl_formal: <none>

 TBL_TYPE   (used 18/25 entries) of OT_TYPE
 Note: '!' after Size - full access requires far pointer.
  Id#  Name              Class       Size        References
 ----- ----------------- ----------- ----  ----------------
  0/ 0  complex           TY_COMPLEX     8
  0/ 1  default           TY_ALIAS       2
         of TBL_TYPE  0/11
  0/ 2  default const     TY_CONST       2        Value: 0
  0/ 3  double            TY_DBL         8
  0/ 4  double complex    TY_DCMPLEX    16
  0/ 5  enumerator        TY_ENMRTR      4
  0/ 6  float             TY_FLT         4
  0/ 7  prototype far     TY_FAR         4
```

```
 0/ 8  prototype near  TY_NEAR        4
 0/ 9  signed char     TY_CHAR        1
 0/10  signed long     TY_LONG        4
 0/11  signed short    TY_SHORT       2
 0/12  unsigned char   TY_CHAR        1
 0/13  unsigned long   TY_LONG        4
 0/14  unsigned short  TY_SHORT       2
 0/15  void            TY_VOID        0
 0/16 int              TY_ALIAS       2
        of TBL_TYPE  0/ 1
 0/17 unsigned         TY_ALIAS       2
        of TBL_TYPE  0/14

TBL_BLOCK  (used 1/1 entries) of OT_BLOCK
 Id # Blk # Name
----- ----- ---------------
 0/ 0     1  external


Block # 1  external          (Level:  1)
  bl_type:   BT_BLOCK
  bl_return: <no return type>
  bl_parent: anchor
  bl_formal: <none>

TBL_TYPE   (used 3/9 entries) of OT_TYPE
Note: '!' after Size - full access requires far pointer.
 Id#  Name              Class      Size        References
----- ----------------- ---------- ----        ----------
 1/ 0  () returning     TY_FUNC       2        TBL_TYPE  0/ 1 ()
 1/ 1  () returning     TY_FUNC       4        TBL_TYPE  1/ 2 ();near/volatile
 1/ 2. near pointer     TY_NEAR       4     to TBL_TYPE  0/ 9 near/volatile

TBL_VAR    (used 8/9 entries) of OT_SYMBOL
 Id#  Name              Usage      Type     Attributes/References
----- ----------------- --------   -----    ---------------------
 1/ 0 calloc            US_DECL     1/ 0
 1/ 1 main              US_DECL     1/ 0
 1/ 2 print2            US_DECL     1/ 0
 1/ 3 printf            US_DECL     1/ 0
 1/ 4 strcat            US_DECL     1/ 1
 1/ 5 strcpy            US_DECL     1/ 1
 1/ 6 sub1              US_DECL     1/ 0
 1/ 7 sub2              US_DECL     1/ 0

TBL_BLOCK  (used 8/9 entries) of OT_BLOCK
 Id # Blk # Name
----- ----- ---------------
 1/ 0     2  main
 1/ 1     3  sub1
 1/ 2     4  sub2
 1/ 3     5  print2
```

```
 1/ 4    6  printf
 1/ 5    7  calloc
 1/ 6    8  strcat
 1/ 7    9  strcpy


Block # 2  main              (Level:  2)
  bl_type:   BT_FUNC
  bl_return:  default
  bl_parent: external
  bl_formal: <none>

TBL_CONST  (used 6/9 entries) of OT_TYPE
Note: 'l' after Size - full access requires far pointer.
 Id#  Name              Class      Size        References
----- ----------------- ---------- ----        ----------
 2/ 0  long constant    TY_CONST      4        Value: 3
 2/ 1  long constant    TY_CONST      4        Value: 4
 2/ 2  long constant    TY_CONST      4        Value: 10
 2/ 3  long constant    TY_CONST      4        Value: 5
 2/ 4  string constant  TY_CONST     23        Value: "Main program finished\n"
 2/ 5  string constant  TY_CONST     22        Value: "Running main program\n"

TBL_CODE   (used 5/9 entries) of OT_TYPE
 Id#  Quadruple Sequence
----- ------------------
 2/ 0     001|00000004 TBL_CODE  2/ 1 TBL_CODE  2/ 2 TBL_CODE  2/ 3 TBL_CODE  2/ 4

 2/ 1     002|00000000 TBL_QUAD  2/ 0 TBL_QUAD  2/ 1

 2/ 2     002|00000001 TBL_QUAD  2/ 2 TBL_QUAD  2/ 3 TBL_QUAD  2/ 4

 2/ 3     002|00000002 TBL_QUAD  2/ 5 TBL_QUAD  2/ 6 TBL_QUAD  2/ 7

 2/ 4     002|00000003 TBL_QUAD  2/ 8 TBL_QUAD  2/ 9


TBL_QUAD   (used 10/17 entries) of OT_QUAD
 Id #  Operation       Left            Right           Third           Result
----- ---------- -------------- -------------- --------------- --------------
 2/ 0       push TBL_TYPE   1/ 2 TBL_CONST  2/ 5  < none >       TBL_TYPE   1/ 2
 2/ 1       call TBL_VAR    1/ 3  < none >        < none >       TBL_TYPE   0/ 1
 2/ 2       push TBL_TYPE   0/16 TBL_CONST  2/ 3  < none >       TBL_TYPE   0/11
 2/ 3       push TBL_TYPE   0/16 TBL_CONST  2/ 0  < none >       TBL_TYPE   0/11
 2/ 4       call TBL_VAR    1/ 6  < none >        < none >       TBL_TYPE   0/ 1
 2/ 5       push TBL_TYPE   0/16 TBL_CONST  2/ 2  < none >       TBL_TYPE   0/11
 2/ 6       push TBL_TYPE   0/16 TBL_CONST  2/ 1  < none >       TBL_TYPE   0/11
 2/ 7       call TBL_VAR    1/ 7  < none >        < none >       TBL_TYPE   0/ 1
 2/ 8       push TBL_TYPE   1/ 2 TBL_CONST  2/ 4  < none >       TBL_TYPE   1/ 2
 2/ 9       call TBL_VAR    1/ 3  < none >        < none >       TBL_TYPE   0/ 1
```

```
Block # 3  sub1              (Level:  2)
  bl_type:   BT_FUNC
  bl_return:  default
  bl_parent: external
  bl_formal:  US_RVALU -1/ 0 US_RVALU -1/ 0

TBL_TYPE   (used 1/1 entries) of OT_TYPE
Note: 'I' after Size - full access requires far pointer.
 Id#  Name              Class      Size        References
-----  ---------------  ---------- ----        ---------------
 3/ 0  [0 ... 99]       TY_ARR      100      of TBL_TYPE  0/ 9 [0 ... 99]

TBL_CONST  (used 5/9 entries) of OT_TYPE
Note: 'I' after Size - full access requires far pointer.
 Id#  Name              Class      Size        References
-----  ---------------  ---------- ----        ---------------
 3/ 0  long constant    TY_CONST      4        Value: 1
 3/ 1  long constant    TY_CONST      4        Value: 2
 3/ 2  long constant    TY_CONST      4        Value: 0
 3/ 3  string constant  TY_CONST     16        Value: "Executing sub1\n"
 3/ 4  upper bound      TY_CONST      4        Value: 99

TBL_VAR    (used 8/9 entries) of OT_SYMBOL
 Id#  Name              Usage      Type      Attributes/References
-----  ---------------  --------   -----     ---------------------
 3/ 0 i                 US_DECL    0/11
 3/ 1 j                 US_DECL    0/11
 3/ 2 k                 US_DECL    0/11
 3/ 3 p1                US_FPARM   0/16
 3/ 4 p2                US_FPARM   0/16
 3/ 5 str               US_DECL    3/ 0
 3/ 6 x                 US_DECL    0/11
 3/ 7 y                 US_DECL    0/11

TBL_CODE   (used 3/9 entries) of OT_TYPE
 Id#  Quadruple Sequence
-----  ------------------
 3/ 0

 3/ 1     001|00000017 TBL_CODE  3/ 2 TBL_QUAD  3/ 3 TBL_QUAD  3/ 6 TBL_QUAD  3/ 9
          TBL_QUAD  3/10 TBL_QUAD  3/11 TBL_QUAD  3/12 TBL_QUAD  3/14
          TBL_QUAD  3/16 TBL_QUAD  3/18 TBL_QUAD  3/19 TBL_QUAD  3/20
          TBL_QUAD  3/21 TBL_QUAD  3/22 TBL_QUAD  3/23 TBL_QUAD  3/24
          TBL_QUAD  3/25 TBL_QUAD  3/26 TBL_QUAD  3/30

 3/ 2     002|00000005 TBL_QUAD  3/ 0 TBL_QUAD  3/ 1


TBL_QUAD   (used 31/33 entries) of OT_QUAD
 Id #  Operation       Left              Right             Third          Result
-----  ----------  ---------------  ---------------  ---------------  ---------------
 3/ 0       push TBL_TYPE   1/ 2 TBL_CONST  3/ 3  < none >       TBL_TYPE   1/ 2
```

```
 3/ 1        call TBL_VAR     1/ 3  < none >          < none >        TBL_TYPE    0/ 1
 3/ 2           = TBL_VAR     3/ 0 TBL_CONST  3/ 2    < none >        TBL_TYPE    0/11
 3/ 3         for TBL_QUAD    3/ 2 TBL_QUAD   3/ 7    < none >         < none >
 3/ 4           < TBL_VAR     3/ 0 TBL_CONST  3/ 0    < none >        TBL_TYPE    0/ 1
 3/ 5        x++ TBL_VAR      3/ 0  < none >          < none >        TBL_TYPE    0/11
 3/ 6       call TBL_BLOCK    3/ 0  < none >          < none >         < none >
 3/ 7      while TBL_QUAD     3/ 4 TBL_QUAD   3/ 5 TBL_QUAD    3/ 6  < none >
 3/ 8         += TBL_VAR      3/ 6 TBL_VAR    3/ 7    < none >        TBL_TYPE    0/11
 3/ 9         -= TBL_VAR      3/ 6 TBL_VAR    3/ 7    < none >        TBL_TYPE    0/11
 3/10         *= TBL_VAR      3/ 6 TBL_VAR    3/ 7    < none >        TBL_TYPE    0/11
 3/11         /= TBL_VAR      3/ 6 TBL_VAR    3/ 7    < none >        TBL_TYPE    0/11
 3/12         &= TBL_VAR      3/ 6 TBL_VAR    3/ 7    < none >        TBL_TYPE    0/11
 3/13          | TBL_VAR      3/ 7  < none >          < none >        TBL_TYPE    0/ 1
 3/14          = TBL_VAR      3/ 6 TBL_QUAD   3/13    < none >        TBL_TYPE    0/11
 3/15          ^ TBL_VAR      3/ 7  < none >          < none >        TBL_TYPE    0/11
 3/16          = TBL_VAR      3/ 6 TBL_QUAD   3/15    < none >        TBL_TYPE    0/11
 3/17         [] TBL_VAR      3/ 5 TBL_CONST  3/ 2    < none >        TBL_TYPE    0/ 9
 3/18          = TBL_VAR      3/ 6 TBL_QUAD   3/17    < none >        TBL_TYPE    0/11
     warning, automatic cast to result type!
 3/19          = TBL_VAR      3/ 6 TBL_VAR    3/ 1    < none >        TBL_TYPE    0/11
 3/20         += TBL_VAR      3/ 6 TBL_VAR    3/ 7    < none >        TBL_TYPE    0/11
 3/21         -= TBL_VAR      3/ 6 TBL_VAR    3/ 7    < none >        TBL_TYPE    0/11
 3/22         *= TBL_VAR      3/ 6 TBL_VAR    3/ 7    < none >        TBL_TYPE    0/11
 3/23         /= TBL_VAR      3/ 6 TBL_VAR    3/ 7    < none >        TBL_TYPE    0/11
 3/24         &= TBL_VAR      3/ 6 TBL_VAR    3/ 7    < none >        TBL_TYPE    0/11
 3/25          = TBL_VAR      3/ 6 TBL_CONST  3/ 1    < none >        TBL_TYPE    0/11
     warning, automatic cast to result type!
 3/26     switch TBL_VAR      3/ 6 TBL_QUAD   3/27 TBL_CODE    3/ 0  < none >
 3/27       call TBL_BLOCK    3/ 1  < none >          < none >         < none >
 3/28          + TBL_VAR      3/ 2 TBL_VAR    3/ 0    < none >        TBL_TYPE    0/11
 3/29          * TBL_VAR      3/ 1 TBL_QUAD   3/28    < none >        TBL_TYPE    0/11
 3/30     return TBL_TYPE     0/ 1 TBL_QUAD   3/29    < none >        TBL_TYPE    0/11

TBL_BLOCK  (used 2/9 entries) of OT_BLOCK
 Id # Blk # Name
----- ----- --------------
 3/ 0   12  ?15
 3/ 1   14  ?16


Block #12  ?15              (Level:  3)
  bl_type:   BT_SUBR
  bl_return: <no return type>
  bl_parent: sub1
  bl_formal: <none>

TBL_CONST  (used 4/9 entries) of OT_TYPE
Note: '!' after Size - full access requires far pointer.
 Id#  Name              Class      Size        Reference:
----- --------------- ---------- ----        ----------
12/ 0  long constant   TY_CONST      4        Value: 1
12/ 1  long constant   TY_CONST      4        Value: 3
```

```
12/ 2  long constant   TY_CONST      4        Value: 0
12/ 3  string constant TY_CONST     13        Value: "loop line 3\n"

TBL_CODE   (used 2/9 entries) of OT_TYPE
 Id#  Quadruple Sequence
----- ------------------
12/ 0     002|00000010 TBL_QUAD 12/ 1 TBL_CODE 12/ 1 TBL_QUAD 12/ 8 TBL_QUAD 12/10
      TBL_QUAD 12/12 TBL_QUAD 12/14 TBL_QUAD 12/15

12/ 1     003|00000009 TBL_QUAD 12/ 6 TBL_QUAD 12/ 7


TBL_QUAD   (used 16/17 entries) of OT_QUAD
 Id #  Operation       Left            Right           Third           Result
----- ---------- -------------- -------------- -------------- --------------
12/ 0          = TBL_VAR    3/ 1 TBL_CONST 12/ 2  < none >       TBL_TYPE   0/11
12/ 1        for TBL_QUAD  12/ 0 TBL_QUAD  12/ 5  < none >        < none >
12/ 2          < TBL_VAR    3/ 1 TBL_CONST 12/ 0  < none >       TBL_TYPE   0/ 1
12/ 3        x++ TBL_VAR    3/ 1  < none >        < none >       TBL_TYPE   0/11
12/ 4       call TBL_BLOCK 12/ 0  < none >        < none >        < none >
12/ 5      while TBL_QUAD  12/ 2 TBL_QUAD  12/ 3 TBL_QUAD  12/ 4  < none >
12/ 6       push TBL_TYPE   1/ 2 TBL_CONST 12/ 3  < none >       TBL_TYPE   1/ 2
12/ 7       call TBL_VAR    1/ 3  < none >        < none >       TBL_TYPE   0/ 1
12/ 8          = TBL_VAR    3/ 7 TBL_CONST 12/ 1  < none >       TBL_TYPE   0/11
      warning, automatic cast to result type!
12/ 9          ! TBL_VAR    3/ 7  < none >        < none >       TBL_TYPE   0/ 1
12/10          = TBL_VAR    3/ 6 TBL_QUAD  12/ 9  < none >       TBL_TYPE   0/11
12/11          ~ TBL_VAR    3/ 7  < none >        < none >       TBL_TYPE   0/11
12/12          = TBL_VAR    3/ 6 TBL_QUAD  12/11  < none >       TBL_TYPE   0/11
12/13         [] TBL_VAR    3/ 5 TBL_CONST 12/ 2  < none >       TBL_TYPE   0/ 9
12/14          = TBL_VAR    3/ 6 TBL_QUAD  12/13  < none >       TBL_TYPE   0/11
      warning, automatic cast to result type!
12/15          = TBL_VAR    3/ 6 TBL_VAR    3/ 1  < none >       TBL_TYPE   0/11

TBL_BLOCK  (used 1/1 entries) of OT_BLOCK
 Id # Blk # Name
----- ----- ---------------
12/ 0   13  ?14


Block #13  ?14              (Level:  4)
  bl_type:   BT_SUBR
  bl_return: <no return type>
  bl_parent: ?15
  bl_formal: <none>

TBL_CONST  (used 4/9 entries) of OT_TYPE
Note: '!' after Size - full access requires far pointer.
 Id#  Name             Class      Size        References
----- --------------- ---------- ----        ----------
13/ 0  long constant   TY_CONST      4        Value: 1
13/ 1  long constant   TY_CONST      4        Value: 0
```

```
13/ 2  string constant TY_CONST     13        Value: "loop line 2\n"
13/ 3  string constant TY_CONST     13        Value: "loop line 1\n"

TBL_CODE   (used 3/9 entries) of OT_TYPE
 Id#  Quadruple Sequence
----- -------------------
13/ 0     003|00000008 TBL_QUAD 13/ 1 TBL_CODE 13/ 1

13/ 1     004|00000007 TBL_QUAD 13/ 7 TBL_QUAD 13/ 8

13/ 2     005|00000006 TBL_QUAD 13/ 4 TBL_QUAD 13/ 5


TBL_QUAD   (used 9/9 entries) of OT_QUAD
 Id #  Operation       Left            Right           Third           Result
----- ---------- -------------- -------------- --------------- ---------------
13/ 0          = TBL_VAR    3/ 2 TBL_CONST 13/ 1  < none >        TBL_TYPE   0/11
13/ 1        for TBL_QUAD  13/ 0 TBL_QUAD  13/ 6  < none >         < none >
13/ 2          < TBL_VAR    3/ 2 TBL_CONST 13/ 0  < none >        TBL_TYPE   0/ 1
13/ 3        x++ TBL_VAR    3/ 2  < none >        < none >        TBL_TYPE   0/11
13/ 4       push TBL_TYPE   1/ 2 TBL_CONST 13/ 3  < none >        TBL_TYPE   1/ 2
13/ 5       call TBL_VAR    1/ 3  < none >        < none >        TBL_TYPE   0/ 1
13/ 6      while TBL_QUAD  13/ 2 TBL_QUAD  13/ 3 TBL_CODE  13/ 2  < none >
13/ 7       push TBL_TYPE   1/ 2 TBL_CONST 13/ 2  < none >        TBL_TYPE   1/ 2
13/ 8       call TBL_VAR    1/ 3  < none >        < none >        TBL_TYPE   0/ 1


Block #14  ?16              (Level:  3)
  bl_type:   BT_SUBR
  bl_return: <no return type>
  bl_parent: sub1
  bl_formal: <none>

TBL_CONST  (used 8/9 entries) of OT_TYPE
Note: '!' after Size - full access requires far pointer.
 Id#  Name              Class      Size        References
----- --------------- ---------- ----        ----------
14/ 0  long constant   TY_CONST      4        Value: 2
14/ 1  long constant   TY_CONST      4        Value: 3
14/ 2  long constant   TY_CONST      4        Value: 1
14/ 3  string constant TY_CONST     16        Value: "last case stmt\n"
14/ 4  string constant TY_CONST     16        Value: "Case 3 of switch"
14/ 5  string constant TY_CONST     12        Value: "default case"
14/ 6  string constant TY_CONST     18        Value: "second case stmt\n"
14/ 7  string constant TY_CONST     17        Value: "first case stmt\n"

TBL_CODE   (used 6/9 entries) of OT_TYPE
 Id#  Quadruple Sequence
----- -------------------
14/ 0     002|00000016 TBL_QUAD 14/ 0 TBL_QUAD 14/ 1 TBL_CODE 14/ 1 TBL_QUAD 14/ 4
      TBL_CODE 14/ 2 TBL_QUAD 14/ 7 TBL_QUAD 14/ 8 TBL_QUAD 14/ 9
      TBL_QUAD 14/10 TBL_CODE 14/ 3 TBL_QUAD 14/14 TBL_QUAD 14/15
```

```
      TBL_CODE 14/ 4 TBL_QUAD 14/20 TBL_CODE 14/ 5

14/ 1     003|00000011 TBL_QUAD 14/ 2 TBL_QUAD 14/ 3

14/ 2     003|00000012 TBL_QUAD 14/ 5 TBL_QUAD 14/ 6

14/ 3     003|00000013 TBL_QUAD 14/11 TBL_QUAD 14/12 TBL_QUAD 14/13

14/ 4     003|00000014 TBL_QUAD 14/16 TBL_QUAD 14/17 TBL_QUAD 14/18

14/ 5     003|00000015 TBL_QUAD 14/21 TBL_QUAD 14/22


TBL_QUAD   (used 23/25 entries) of OT_QUAD
 Id #  Operation       Left           Right          Third          Result
------ ---------- -------------- -------------- -------------- --------------
14/ 0       case TBL_CONST 14/ 2  < none >        < none >        < none >
14/ 1          = TBL_VAR    3/ 7 TBL_CONST 14/ 2  < none >       TBL_TYPE   0/11
     warning, automatic cast to result type!
14/ 2       push TBL_TYPE   1/ 2 TBL_CONST 14/ 7  < none >       TBL_TYPE   1/ 2
14/ 3       call TBL_VAR    1/ 3  < none >        < none >       TBL_TYPE   0/ 1
14/ 4       case TBL_CONST 14/ 0  < none >        < none >        < none >
14/ 5       push TBL_TYPE   1/ 2 TBL_CONST 14/ 6  < none >       TBL_TYPE   1/ 2
14/ 6       call TBL_VAR    1/ 3  < none >        < none >       TBL_TYPE   0/ 1
14/ 7          = TBL_VAR    3/ 7 TBL_CONST 14/ 0  < none >       TBL_TYPE   0/11
     warning, automatic cast to result type!
14/ 8      break TBL_QUAD   3/26  < none >        < none >        < none >
14/ 9       case TBL_CONST 14/ 1  < none >        < none >        < none >
14/10          = TBL_VAR    3/ 6 TBL_CONST 14/ 1  < none >       TBL_TYPE   0/11
     warning, automatic cast to result type!
14/11       push TBL_TYPE   1/ 2 TBL_VAR    3/ 5  < none >       TBL_TYPE   1/ 2
     warning, automatic cast to result type!
14/12       push TBL_TYPE   1/ 2 TBL_CONST 14/ 4  < none >       TBL_TYPE   1/ 2
14/13       call TBL_VAR    1/ 5  < none >        < none >       TBL_TYPE   1/ 2
14/14      break TBL_QUAD   3/26  < none >        < none >        < none >
14/15       case  < none >        < none >        < none >        < none >
14/16       push TBL_TYPE   1/ 2 TBL_VAR    3/ 5  < none >       TBL_TYPE   1/ 2
     warning, automatic cast to result type!
14/17       push TBL_TYPE   1/ 2 TBL_CONST 14/ 5  < none >       TBL_TYPE   1/ 2
14/18       call TBL_VAR    1/ 5  < none >        < none >       TBL_TYPE   1/ 2
14/19          + TBL_VAR    3/ 6 TBL_CONST 14/ 2  < none >       TBL_TYPE   0/10
     warning, automatic cast to result type!
14/20          = TBL_VAR    3/ 7 TBL_QUAD  14/19  < none >       TBL_TYPE   0/11
     warning, automatic cast to result type!
14/21       push TBL_TYPE   1/ 2 TBL_CONST 14/ 3  < none >       TBL_TYPE   1/ 2
14/22       call TBL_VAR    1/ 3  < none >        < none >       TBL_TYPE   0/ 1


Block # 4  sub2              (Level:  2)
  bl_type:   BT_FUNC
  bl_return:  default
  bl_parent: external
```

```
  bl_formal:  US_RVALU -1/ 0 US_RVALU -1/ 0

TBL_CONST  (used 10/17 entries) of OT_TYPE
Note: '!' after Size - full access requires far pointer.
 Id#  Name              Class      Size        References
----- ----------------- ---------- ----        ----------
 4/ 0  char constant    TY_CONST      1        Value: '\0'
 4/ 1  long constant    TY_CONST      4        Value: 5
 4/ 2  long constant    TY_CONST      4        Value: 20
 4/ 3  long constant    TY_CONST      4        Value: 1
 4/ 4  long constant    TY_CONST      4        Value: 10
 4/ 5  long constant    TY_CONST      4        Value: 0
 4/ 6  string constant  TY_CONST      4        Value: "prt1"
 4/ 7  string constant  TY_CONST      5        Value: " cat1"
 4/ 8  string constant  TY_CONST     10        Value: "first copy"
 4/ 9  string constant  TY_CONST     16        Value: "Executing sub2\n"

TBL_VAR    (used 8/9 entries) of OT_SYMBOL
 Id#  Name              Usage      Type      Attributes/References
----- ----------------- --------   -----     ---------------------
 4/ 0 i                 US_DECL     0/10
 4/ 1 j                 US_DECL     0/10
 4/ 2 k                 US_DECL     0/16
 4/ 3 param1            US_FPARM    0/16
 4/ 4 param2            US_FPARM    0/16
 4/ 5 str1              US_DECL     1/ 2
 4/ 6 str2              US_DECL     1/ 2
 4/ 7 str3              US_DECL     1/ 2

TBL_CODE   (used 9/9 entries) of OT_TYPE
 Id#  Quadruple Sequence
----- -------------------
 4/ 0     001|00000031 TBL_CODE  4/ 1 TBL_QUAD  4/ 3 TBL_QUAD  4/14 TBL_QUAD  4/19
      TBL_QUAD  4/24 TBL_CODE  4/ 5 TBL_QUAD  4/29 TBL_CODE  4/ 6

 4/ 1     002|00000018 TBL_QUAD  4/ 0 TBL_QUAD  4/ 1

 4/ 2     002|00000024 TBL_QUAD  4/10 TBL_QUAD  4/11 TBL_QUAD  4/12

 4/ 3     002|00000025 TBL_QUAD  4/15 TBL_QUAD  4/16 TBL_QUAD  4/17

 4/ 4     002|00000026 TBL_QUAD  4/20 TBL_QUAD  4/21 TBL_QUAD  4/22

 4/ 5     002|00000027 TBL_QUAD  4/25 TBL_QUAD  4/26 TBL_QUAD  4/27

 4/ 6     002|00000030 TBL_QUAD  4/36 TBL_QUAD  4/37 TBL_QUAD  4/38 TBL_QUAD  4/39

 4/ 7     003|00000029 TBL_QUAD  4/33 TBL_QUAD  4/34 TBL_QUAD  4/35

 4/ 8     004|00000028 TBL_QUAD  4/30 TBL_QUAD  4/31 TBL_QUAD  4/32
```

```
TBL_QUAD   (used 40/41 entries) of OT_QUAD
 Id #  Operation       Left            Right           Third           Result
-----  ----------  --------------  --------------  --------------  --------------
 4/ 0        push TBL_TYPE   1/ 2 TBL_CONST  4/ 9  < none >        TBL_TYPE   1/ 2
 4/ 1        call TBL_VAR    1/ 3  < none >         < none >        TBL_TYPE   0/ 1
 4/ 2           = TBL_VAR    4/ 0 TBL_CONST  4/ 5  < none >        TBL_TYPE   0/10
 4/ 3         for TBL_QUAD   4/ 2 TBL_QUAD   4/ 9  < none >         < none >
 4/ 4           < TBL_VAR    4/ 0 TBL_CONST  4/ 1  < none >        TBL_TYPE   0/ 1
 4/ 5         x++ TBL_VAR    4/ 0  < none >         < none >        TBL_TYPE   0/10
 4/ 6          do TBL_QUAD   4/ 7  < none >       TBL_QUAD   4/ 8  < none >
 4/ 7        call TBL_BLOCK  4/ 0  < none >         < none >         < none >
 4/ 8           < TBL_VAR    4/ 0 TBL_CONST  4/ 4  < none >        TBL_TYPE   0/ 1
 4/ 9       while TBL_QUAD   4/ 4 TBL_QUAD   4/ 5 TBL_QUAD   4/ 6  < none >
 4/10        push TBL_TYPE   0/16 TBL_CONST  4/ 2  < none >        TBL_TYPE   0/11
 4/11        push TBL_TYPE   0/16 TBL_CONST  4/ 3  < none >        TBL_TYPE   0/11
 4/12        call TBL_VAR    1/ 0  < none >         < none >        TBL_TYPE   0/ 1
 4/13      (cast) TBL_TYPE   1/ 2 TBL_CODE   4/ 2  < none >        TBL_TYPE   1/ 2
 4/14           = TBL_VAR    4/ 5 TBL_QUAD   4/13  < none >        TBL_TYPE   1/ 2
 4/15        push TBL_TYPE   0/16 TBL_CONST  4/ 2  < none >        TBL_TYPE   0/11
 4/16        push TBL_TYPE   0/16 TBL_CONST  4/ 3  < none >        TBL_TYPE   0/11
 4/17        call TBL_VAR    1/ 0  < none >         < none >        TBL_TYPE   0/ 1
 4/18      (cast) TBL_TYPE   1/ 2 TBL_CODE   4/ 3  < none >        TBL_TYPE   1/ 2
 4/19           = TBL_VAR    4/ 6 TBL_QUAD   4/18  < none >        TBL_TYPE   1/ 2
 4/20        push TBL_TYPE   0/16 TBL_CONST  4/ 2  < none >        TBL_TYPE   0/11
 4/21        push TBL_TYPE   0/16 TBL_CONST  4/ 3  < none >        TBL_TYPE   0/11
 4/22        call TBL_VAR    1/ 0  < none >         < none >        TBL_TYPE   0/ 1
 4/23      (cast) TBL_TYPE   1/ 2 TBL_CODE   4/ 4  < none >        TBL_TYPE   1/ 2
 4/24           = TBL_VAR    4/ 7 TBL_QUAD   4/23  < none >        TBL_TYPE   1/ 2
 4/25        push TBL_TYPE   1/ 2 TBL_VAR    4/ 5  < none >        TBL_TYPE   1/ 2
 4/26        push TBL_TYPE   1/ 2 TBL_CONST  4/ 8  < none >        TBL_TYPE   1/ 2
 4/27        call TBL_VAR    1/ 5  < none >         < none >        TBL_TYPE   1/ 2
 4/28          [] TBL_VAR    4/ 6 TBL_CONST  4/ 5  < none >        TBL_TYPE   0/ 9
 4/29           = TBL_QUAD   4/28 TBL_CONST  4/ 0  < none >        TBL_TYPE   0/ 9
 4/30        push TBL_TYPE   1/ 2 TBL_VAR    4/ 7  < none >        TBL_TYPE   1/ 2
 4/31        push TBL_TYPE   1/ 2 TBL_CONST  4/ 7  < none >        TBL_TYPE   1/ 2
 4/32        call TBL_VAR    1/ 5  < none >         < none >        TBL_TYPE   1/ 2
 4/33        push TBL_TYPE   1/ 2 TBL_VAR    4/ 6  < none >        TBL_TYPE   1/ 2
 4/34        push TBL_TYPE   1/ 2 TBL_CODE   4/ 8  < none >        TBL_TYPE   1/ 2
 4/35        call TBL_VAR    1/ 4  < none >         < none >        TBL_TYPE   1/ 2
 4/36        push TBL_TYPE   1/ 2 TBL_VAR    4/ 5  < none >        TBL_TYPE   1/ 2
 4/37        push TBL_TYPE   1/ 2 TBL_CODE   4/ 7  < none >        TBL_TYPE   1/ 2
 4/38        push TBL_TYPE   1/ 2 TBL_CONST  4/ 6  < none >        TBL_TYPE   1/ 2
 4/39        call TBL_VAR    1/ 2  < none >         < none >        TBL_TYPE   0/ 1

TBL_BLOCK  (used 1/1 entries) of OT_BLOCK
 Id # Blk # Name
----- ----- ---------------
 4/ 0   16  ?19


Block #16  ?19              (Level:  3)
  bl_type:   BT_SUBR
```

```
  bl_return: <no return type>
  bl_parent: sub2
  bl_formal: <none>

TBL_CONST  (used 3/9 entries) of OT_TYPE
Note: '!' after Size - full access requires far pointer.
 Id#  Name             Class      Size        References
----- ---------------- ---------- ----        ----------
16/ 0  long constant   TY_CONST      4        Value: 130
16/ 1  long constant   TY_CONST      4        Value: 30
16/ 2  string constant TY_CONST     11        Value: "End of DO\n"

TBL_CODE   (used 2/9 entries) of OT_TYPE
 Id#  Quadruple Sequence
----- -------------------
16/ 0     003|00000023 TBL_QUAD 16/ 0 TBL_QUAD 16/ 2 TBL_QUAD 16/ 4 TBL_CODE 16/ 1

16/ 1     004|00000022 TBL_QUAD 16/ 5 TBL_QUAD 16/ 6


TBL_QUAD   (used 7/9 entries) of OT_QUAD
 Id #  Operation       Left            Right           Third           Result
----- ---------- --------------- --------------- --------------- ---------------
16/ 0          = TBL_VAR    4/ 1 TBL_CONST 16/ 1  < none >        TBL_TYPE   0/10
16/ 1         <= TBL_VAR    4/ 1 TBL_CONST 16/ 0  < none >        TBL_TYPE   0/ 1
16/ 2      while TBL_QUAD  16/ 1  < none >        TBL_QUAD  16/ 3  < none >
16/ 3       call TBL_BLOCK 16/ 0  < none >         < none >        < none >
16/ 4        x++ TBL_VAR    4/ 0  < none >         < none >        TBL_TYPE   0/10
16/ 5       push TBL_TYPE   1/ 2 TBL_CONST 16/ 2  < none >        TBL_TYPE   1/ 2
16/ 6       call TBL_VAR    1/ 3  < none >         < none >        TBL_TYPE   0/ 1

TBL_BLOCK  (used 1/1 entries) of OT_BLOCK
 Id # Blk # Name
----- ----- ---------------
16/ 0   17  ?18


Block #17  ?18              (Level:  4)
  bl_type:   BT_SUBR
  bl_return: <no return type>
  bl_parent: ?19
  bl_formal: <none>

TBL_CONST  (used 3/9 entries) of OT_TYPE
Note: '!' after Size - full access requires far pointer.
 Id#  Name             Class      Size        References
----- ---------------- ---------- ----        ----------
17/ 0  long constant   TY_CONST      4        Value: 10
17/ 1  long constant   TY_CONST      4        Value: 100
17/ 2  long constant   TY_CONST      4        Value: 0

TBL_CODE   (used 1/1 entries) of OT_TYPE
```

```
 Id#  Quadruple Sequence
----- --------------------
17/ 0      004|00000021 TBL_QUAD 17/ 1 TBL_QUAD 17/ 5


TBL_QUAD   (used 6/9 entries) of OT_QUAD
 Id #  Operation       Left            Right           Third           Result
----- ---------- -------------- -------------- --------------- --------------
17/ 0          = TBL_VAR    4/ 2 TBL_CONST 17/ 2  < none >        TBL_TYPE   0/11
17/ 1        for TBL_QUAD  17/ 0 TBL_QUAD  17/ 4  < none >         < none >
17/ 2          < TBL_VAR    4/ 2 TBL_CONST 17/ 1  < none >        TBL_TYPE   0/ 1
17/ 3       call TBL_BLOCK 17/ 0  < none >         < none >         < none >
17/ 4      while TBL_QUAD  17/ 2  < none >       TBL_QUAD  17/ 3  < none >
17/ 5         += TBL_VAR    4/ 1 TBL_CONST 17/ 0  < none >        TBL_TYPE   0/10

TBL_BLOCK  (used 1/1 entries) of OT_BLOCK
 Id # Blk # Name
----- ----- ---------------
17/ 0   18  ?17


Block #18  ?17              (Level:  5)
  bl_type:   BT_SUBR
  bl_return: <no return type>
  bl_parent: ?18
  bl_formal: <none>

TBL_CONST  (used 4/9 entries) of OT_TYPE
Note: 'l' after Size - full access requires far pointer.
 Id#  Name             Class      Size        References
----- --------------- ---------- ----        ----------------
18/ 0  long constant   TY_CONST      4        Value: 10
18/ 1  long constant   TY_CONST      4        Value: 50
18/ 2  long constant   TY_CONST      4        Value: 2
18/ 3  string constant TY_CONST     17        Value: "inner loop line\n"

TBL_CODE   (used 2/9 entries) of OT_TYPE
 Id#  Quadruple Sequence
----- --------------------
18/ 0      005|00000020 TBL_CODE 18/ 1 TBL_QUAD 18/ 5 TBL_QUAD 18/ 6

18/ 1      006|00000019 TBL_QUAD 18/ 0 TBL_QUAD 18/ 1


TBL_QUAD   (used 7/9 entries) of OT_QUAD
 Id #  Operation       Left            Right           Third           Result
----- ---------- -------------- -------------- --------------- --------------
18/ 0       push TBL_TYPE   1/ 2 TBL_CONST 18/ 3  < none >        TBL_TYPE   1/ 2
18/ 1       call TBL_VAR    1/ 3  < none >         < none >        TBL_TYPE   0/ 1
18/ 2          * TBL_VAR    4/ 1 TBL_CONST 18/ 2  < none >        TBL_TYPE   0/10
18/ 3          + TBL_VAR    4/ 0 TBL_CONST 18/ 0  < none >        TBL_TYPE   0/10
18/ 4          + TBL_QUAD  18/ 2 TBL_QUAD  18/ 3  < none >        TBL_TYPE   0/10
```

```
18/ 5          = TBL_VAR    4/ 2 TBL_QUAD  18/ 4  < none >       TBL_TYPE   0/11
     warning, automatic cast to result type!
18/ 6         += TBL_VAR    4/ 1 TBL_CONST 18/ 1  < none >       TBL_TYPE   0/10


Block # 5  print2          (Level:  2)
  bl_type:   BT_FUNC
  bl_return:  default
  bl_parent: external
  bl_formal:  US_RVALU -1/ 0 US_RVALU -1/ 0 US_RVALU -1/ 0

TBL_CONST  (used 1/1 entries) of OT_TYPE
Note: 'l' after Size - full access requires far pointer.
 Id#  Name              Class      Size        References
----- ----------------- ---------- ----        ----------
 5/ 0  string constant TY_CONST      2        Value: "\n"

TBL_VAR    (used 3/9 entries) of OT_SYMBOL
 Id#  Name              Usage     Type     Attributes/References
----- ----------------- --------  -----    ---------------------
 5/ 0 str1              US_FPARM   1/ 2
 5/ 1 str2              US_FPARM   1/ 2
 5/ 2 str3              US_FPARM   1/ 2

TBL_CODE   (used 5/9 entries) of OT_TYPE
 Id#  Quadruple Sequence
----- -------------------
 5/ 0     001¦00000036 TBL_CODE  5/ 1 TBL_CODE  5/ 2 TBL_CODE  5/ 3 TBL_CODE  5/ 4

 5/ 1     002¦00000032 TBL_QUAD  5/ 0 TBL_QUAD  5/ 1

 5/ 2     002¦00000033 TBL_QUAD  5/ 2 TBL_QUAD  5/ 3

 5/ 3     002¦00000034 TBL_QUAD  5/ 4 TBL_QUAD  5/ 5

 5/ 4     002¦00000035 TBL_QUAD  5/ 6 TBL_QUAD  5/ 7


TBL_QUAD   (used 8/9 entries) of OT_QUAD
 Id #  Operation       Left            Right           Third          Result
----- ---------- -------------- -------------- -------------- --------------
 5/ 0       push TBL_TYPE   1/ 2 TBL_VAR    5/ 0  < none >       TBL_TYPE   1/ 2
 5/ 1       call TBL_VAR    1/ 3  < none >        < none >       TBL_TYPE   0/ 1
 5/ 2       push TBL_TYPE   1/ 2 TBL_VAR    5/ 1  < none >       TBL_TYPE   1/ 2
 5/ 3       call TBL_VAR    1/ 3  < none >        < none >       TBL_TYPE   0/ 1
 5/ 4       push TBL_TYPE   1/ 2 TBL_VAR    5/ 2  < none >       TBL_TYPE   1/ 2
 5/ 5       call TBL_VAR    1/ 3  < none >        < none >       TBL_TYPE   0/ 1
 5/ 6       push TBL_TYPE   1/ 2 TBL_CONST  5/ 0  < none >       TBL_TYPE   1/ 2
 5/ 7       call TBL_VAR    1/ 3  < none >        < none >       TBL_TYPE   0/ 1


Block # 6  printf          (Level:  2)
```

```
  bl_type:   BT_PROTO
  bl_return:  default
  bl_parent: external
  bl_formal:  TBL_VAR  6/ 0

TBL_VAR    (used 1/1 entries) of OT_SYMBOL
 Id#  Name              Usage     Type     Attributes/References
----- ----------------- --------  -----    ---------------------
 6/ 0 ?7                US_FPARM   1/ 2


Block # 7  calloc           (Level:  2)
  bl_type:   BT_PROTO
  bl_return:  default
  bl_parent: external
  bl_formal:  TBL_VAR  7/ 0 TBL_VAR  7/ 1

TBL_VAR    (used 2/9 entries) of OT_SYMBOL
 Id#  Name              Usage     Type     Attributes/References
----- ----------------- --------  -----    ---------------------
 7/ 0 ?8                US_FPARM   0/16
 7/ 1 ?9                US_FPARM   0/16


Block # 8  strcat           (Level:  2)
  bl_type:   BT_PROTO
  bl_return: near/volatile
  bl_parent: external
  bl_formal:  TBL_VAR  8/ 0 TBL_VAR  8/ 1

TBL_VAR    (used 2/9 entries) of OT_SYMBOL
 Id#  Name              Usage     Type     Attributes/References
----- ----------------- --------  -----    ---------------------
 8/ 0 ?10               US_FPARM   1/ 2
 8/ 1 ?11               US_FPARM   1/ 2


Block # 9  strcpy           (Level:  2)
  bl_type:   BT_PROTO
  bl_return: near/volatile
  bl_parent: external
  bl_formal:  TBL_VAR  9/ 0 TBL_VAR  9/ 1

TBL_VAR    (used 2/9 entries) of OT_SYMBOL
 Id#  Name              Usage     Type     Attributes/References
----- ----------------- --------  -----    ---------------------
 9/ 0 ?12               US_FPARM   1/ 2
 9/ 1 ?13               US_FPARM   1/ 2
```

MAIN NASSI-SHNIDERMANN LANGUAGE REPRESENATION

```
subroutine
  main,
  return_type  default,
  called_by
    ' ',
  calls
    printf,
    sub1,
    sub2,
  params
    ' ',
  ext_vars
    ' ',
  loc_vars
    ' ',
  metrics
    ' ',
(
  printf("Running main program\n");
  sub1(5, 3);
  sub2(10, 4);
  printf("Main program finished\n");
)

subroutine
  sub1,
  return_type  default,
  called_by
    ' ',
  calls
    printf,
    strcpy,
  params
    ' signed short p1',
    ' signed short p2',
  ext_vars
    ' ',
  loc_vars
    ' signed short i',
    ' signed short j',
    ' signed short k',
    ' signed char str[100]',
    ' signed short x',
    ' signed short y',
  metrics
    ' ',
(
  printf("Executing sub1\n");
  i = 0;
  while (i < 1)
    (
      j = 0;
      while (j < 1)
        (
          page_break P1,
        )
      page_break P2,
    )
  page_break P3,
)

subroutine
  sub2,
  return_type  default,
  called_by
    ' ',
  calls
    printf,
    calloc,
    strcpy,
    strcat,
    print2,
  params
    ' signed short param1',
    ' signed short param2',
  ext_vars
    ' ',
  loc_vars
    ' signed long i',
    ' signed long j',
    ' signed short k',
    ' signed char  *str1',
    ' signed char  *str2',
    ' signed char  *str3',
  metrics
    ' ',
(
  printf("Executing sub2\n");
  i = 0;
  while (i < 5)
    (
      page_break P6,
    )
  str1 = ( signed char  *)calloc(20, 1);
  str2 = ( signed char  *)calloc(20, 1);
  page_break P7,
)

subroutine
  print2,
  return_type  default,
  called_by
    ' ',
```

```
  calls
    printf,
  params
    ' signed char  *str1',
    ' signed char  *str2',
    ' signed char  *str3',
  ext_vars
    ' ',
  loc_vars
    ' ',
  metrics
    ' ',
{
  printf(str1);
  printf(str2);
  printf(str3);
  printf("\n");
}

page P1,
{
  k = 0;
  while (k < 1)
    {
      printf("loop line 1\n");
      k++;
    }
  printf("loop line 2\n");
  j++;
}

page P2,
{
  printf("loop line 3\n");
  y = 3;
  x = !y;
  x = ~y;
  x = str[0];
  x = j;
  i++;
}

page P3,
{
  x += y;
  x -= y;
  x *= y;
  x /= y;
  x &= y;
  x = !y;
  x = ~y;
  x = str[0];
  x = j;
  x += y;
  x -= y;
  x *= y;
  x /= y;
  x &= y;
  x = 2;
  case (x)
    {
      (1) :
        y = 1;
        printf("first case stmt\n");
      (2) :
        printf("second case stmt\n");
        y = 2;
      break
        page_break P4,
      }
    return (j * (k + i));
  }

page P4,
{
  case_cont
    {
      (3) :
        x = 3;
        strcpy(str, "Case 3 of switch");
      break
        default :
        strcpy(str, "default case");
        y = x + 1;
        printf("last case stmt\n");
    }
}

page P6,
{
  do
    {
      j = 30;
      while (j <= 130)
        {
          k = 0;
          while (k < 100)
            {
              printf("inner loop line\n");
              k = j * 2 + (i + 10);
              j += 50;
            }
          j += 10;
```

```
        )
      i++;
      printf("End of DO\n");
    )
  until (i < 10)
  i++;
)

page P7,
{
  str3 = ( signed char  *)calloc(20, 1);
  strcpy(str1, "first copy");
  str2[0] =  \0;
  print2(str1, strcat(str2, strcpy(str3, " cat1")), "prt1");
}
```

THE RESULTING NASSI-SCHEIDERMANN CHART

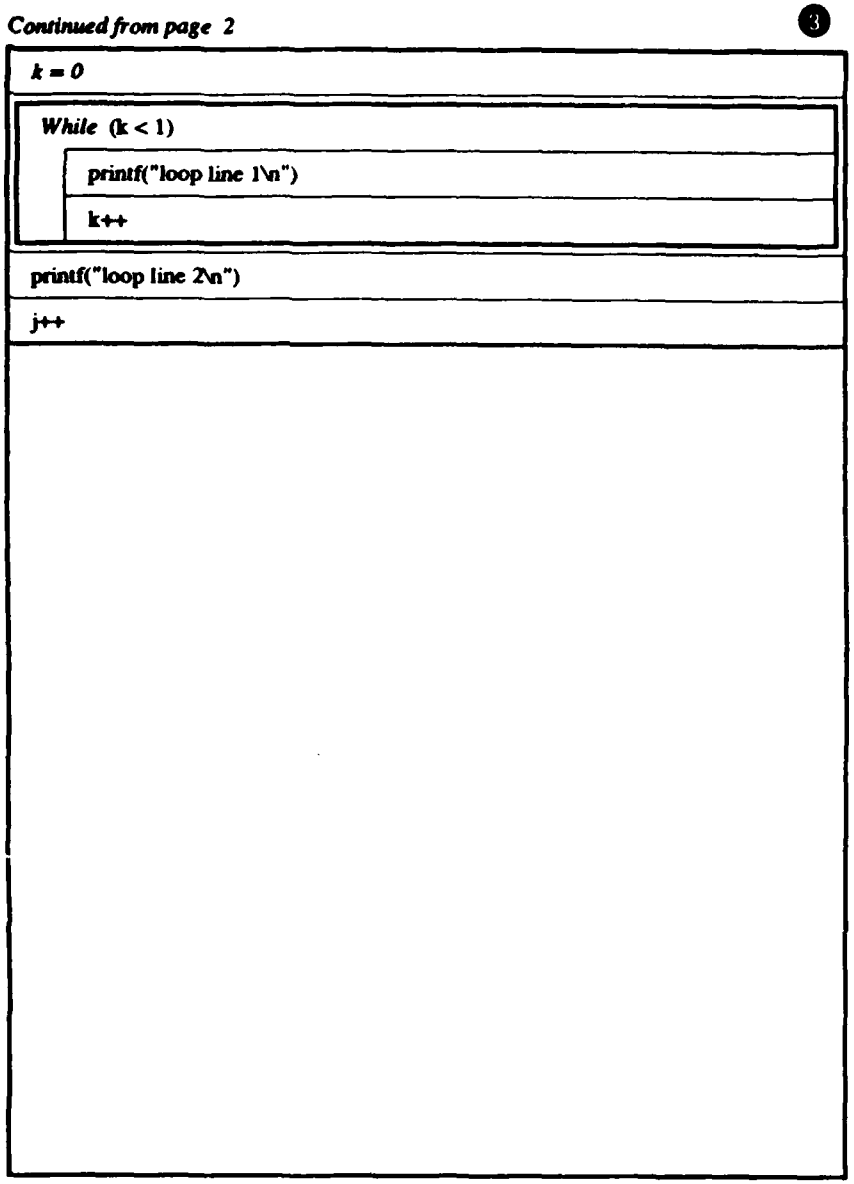
❶

main	*Return Type* default
Called By	*Calls* printf sub1 sub2
Parameters	*Metrics*
External Variables Affected	
Local Variables	
printf("Running main program\n")	
sub1(5, 3)	
sub2(10, 4)	
printf("Main program finished\n")	

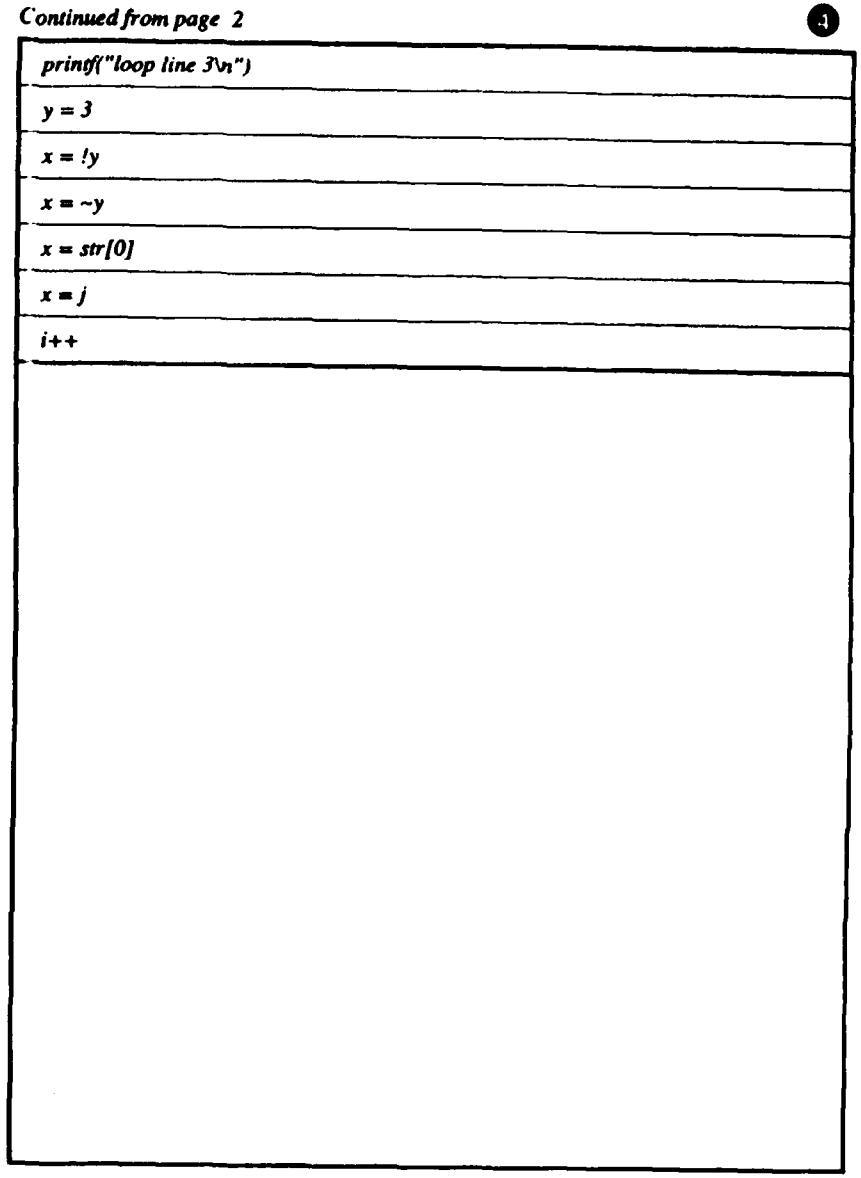
❷

sub1	*Return Type* default
Called By	*Calls* printf strcpy
Parameters signed short p1 signed short p2	*Metrics*
External Variables Affected	
Local Variables signed short i signed short j signed short k signed char str[100] signed short x signed short y	
printf("Executing sub1\n")	
i = 0	
While (i < 1)	
j = 0	
While (j < 1)	
❸	
❹	
❺	

Continued from page 2



Continued from page 2



Continued from page 2

5

x += y
x -= y
*x *= y*
x /= y
x &= y
x = !y
x = ~y
x = str[0]
x = j
x += y
x -= y
*x *= y*
x /= y
x &= y
x = 2

Case of (x)	
(1)	y = 1
	printf("first case stmt\n")
(2)	printf("second case stmt\n")
	y = 2
	10

return (j * (k + i))

6

sub2	*Return Type* default
Called By	*Calls* printf calloc strcpy strcat print2
Parameters signed short param1 signed short param2	*Metrics*
External Variables Affected	
Local Variables signed long i signed long j signed short k signed char *str1 signed char *str2 signed char *str3	

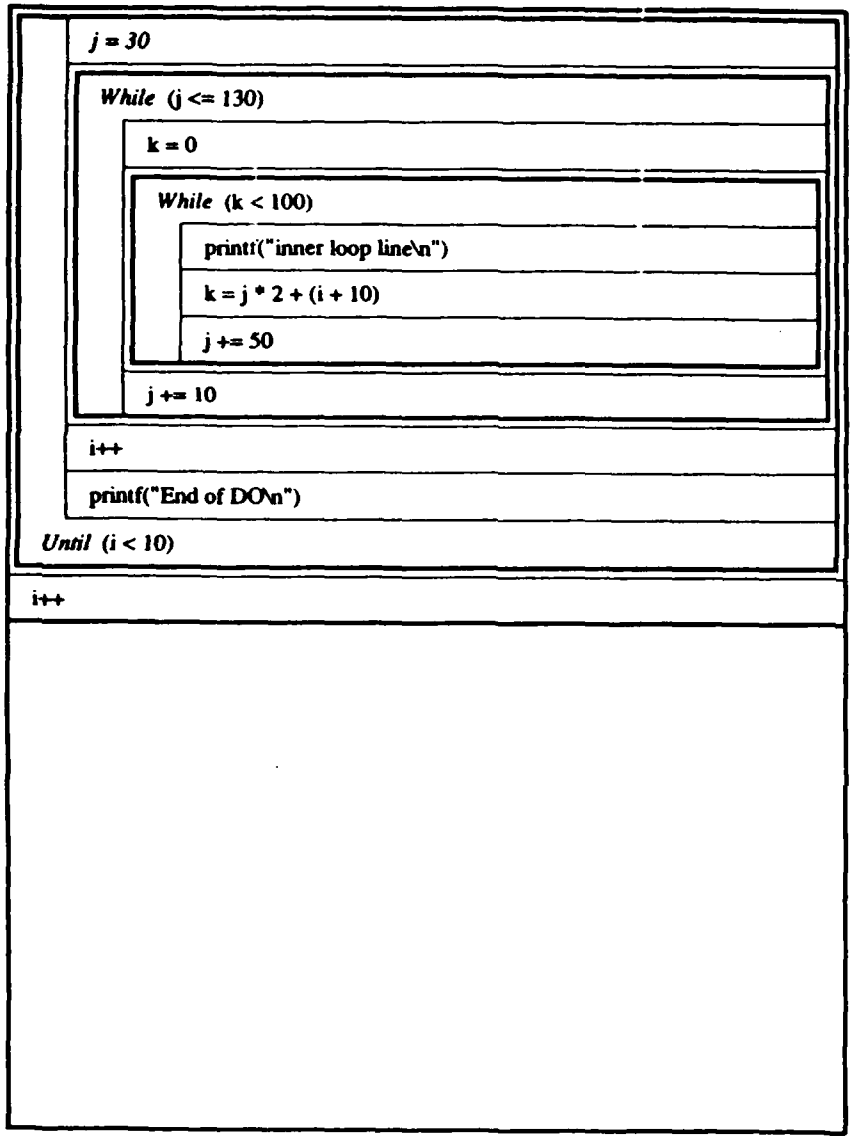
printf("Executing sub2\n")
i = 0

While (i < 5)
7

str1 = (signed char *)calloc(20, 1)
str2 = (signed char *)calloc(20, 1)
8

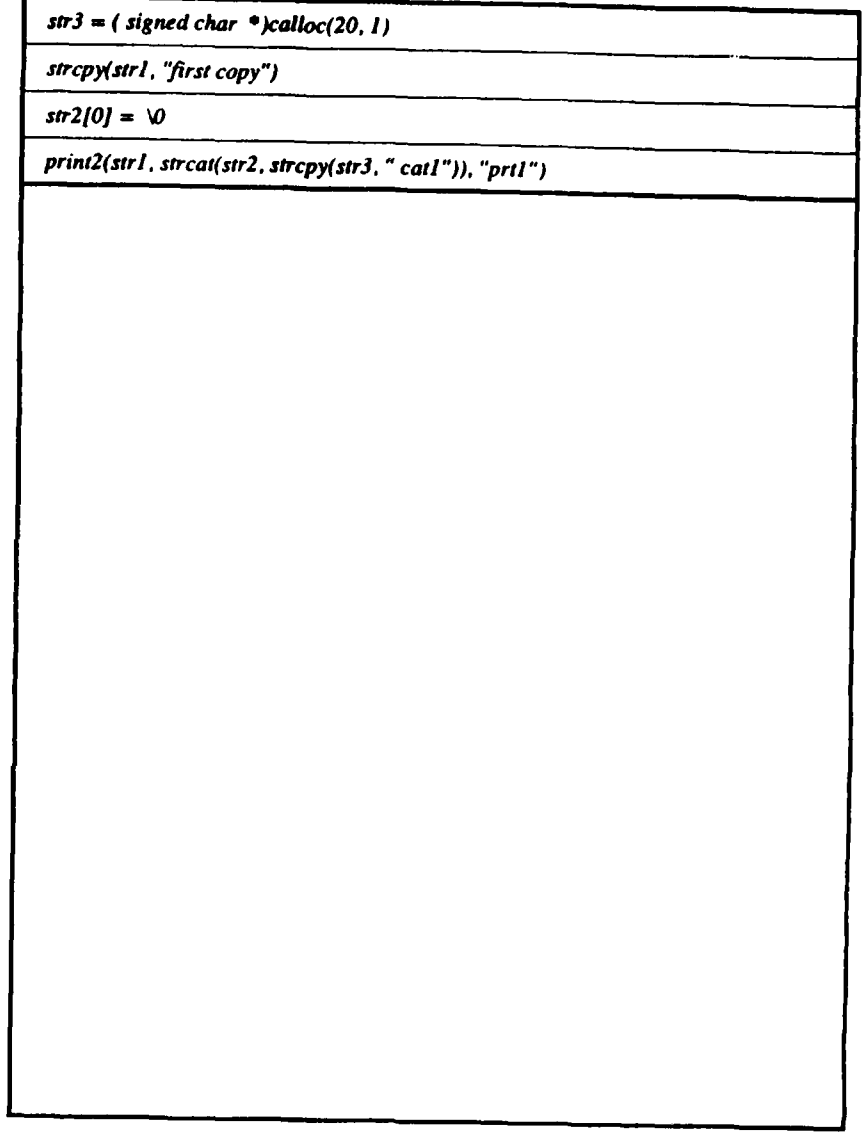
Continued from page 6

7



Continued from page 6

8



9

print2	*Return Type* default
Called By	*Calls* printf
Parameters signed char *str1 signed char *str2 signed char *str3	*Metrics*
External Variables Affected	
Local Variables	

printf(str1)

printf(str2)

printf(str3)

printf("\n")

Continued from page 5

10

Case continued	
(3)	x = 3
	strcpy(str, "Case 3 of switch")
Default	strcpy(str, "default case")
	y = x + 1
	printf("last case stmt\n")

APPENDIX D

RELATED READING

[ADA] Ada Joint Program Office, *Reference Manual for the Ada Programming Language*, (United States Department of Defense, Washington, DC, 1983).

[Aho75] A.V. Aho and M.J. Corasick, "Efficient String Matching: An Aid to Bibliographic Search," *Communications of the ACM*, Vol.18, 1975.

[Aho77] A.V. Aho and J.D. Ullman, *Principles of Compiler Design*, (Addison-Wesley, Reading, MA, 1977).

[Aho83] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *Data Structures* and Algorithms, (Addison- Wesley, Reading, MA, 1983).

[Aho86] A.V. Aho, R. Sethi and J.D. Ullman, *Compilers, Principles, Techniques*, and Tools, (Addison-Wesley, Reading, MA, 1986).

[Arang] G. Arango, I. Baxter, P. Freeman and C. Pidgeon, "Maintaining and Porting of Software by Design Recovery," *Proceedings of the Conference on Software Maintenance* (CSM-85), (Washington, DC, November 1985).

[Barre] W.A. Barrett and J.D. Couch, *Compiler Construction: Theory and Practice*, (Science Research Associates, Chicago, 1979).

[Berg] H.K. Berg, W.E. Boebert, W.R. Franta and T.G. Moher, *Formal Methods of Program Verification and Specification*, (Prentice-Hall, NJ, 1982).

[Biggs] C.L. Biggs, E.G. Birks and W. Atkins, *Managing the Systems Development Process*, (Prentice-Hall, NJ, 1980).

[Birre] N.D. Birrell and M.A. Ould, *A Practical Handbook for Software Development*, (Cambridge University Press, NY, 1985).

[Boar] B.H. Boar, *Application Prototyping*, (John Wiley & Sons, NY, 1984).

[Brons] G.M. Bronstein and R.I. Okamoto, "I'm OK, You're OK, Maintenance is OK," *Tutorial on Software Maintenance*, G. Parikh and N. Zvegintzov, eds., (IEEE Computer Society Press, Silver Spring, MD, 1983).

[Brown] P.J. Brown, "Why Does Software Die?," *Tutorial on Software Maintenance*, G. Parikh and N. Zvegintzov, eds., (IEEE Computer Society Press, Silver Spring, MD, 1983).

[Bush] E. Bush, "The Automatic Restructuring of COBOL," Proceedings of the Conference on Software Maintenance (CSM-85), (Washington, DC, November 1985).

[Calin] P. Calingaert, *Assemblers, Compilers, and Program Translation,* (Computer Science Press, MD, 1979).

[Calli] F.W. Calliss, M. Khalil, M. Munro and M. Ward, "A Knowledge-Based System for Software Maintenance," *Proceedings of the Conference on Software Maintenance* (CSM- 88), (Phoenix, AZ, October, 1988).

[DeMar] T. DeMarco, *Controlling Software Projects*, (Yourdon Press, NY, 1982).

[Fay] S.D. Fay and D.G. Holmes, "Help! I Have to Update an Undocumented Program," *Proceedings of the Conference on Software Maintenance* (CSM-85), (Washington, DC, November 1985).

[Feuer] A.R. Feuer, *The C Puzzle Book*, (Prentice-Hall, NJ, 1982).

[Fjeldst] R.K. Fjeldstad and W.T. Hamlin, "Application Program Maintenance Study - Report to Our Respondents," *Tutorial on Software Maintenance*, G. Parikh and N. Zvegintzov, eds., (IEEE Computer Society Press, Silver Spring, MD, 1983).

[Flett] N.T. Fletton and M. Munro, "Redocumenting Software Systems Using Hypertext Technology," *Proceedings of the Conference on Software Maintenance* (CSM-88), (Phoenix, AZ, October, 1988).

[Foste] J.R. Foster and M. Munro, "A Documentation Method Based on Cross-Referencing," *Proceedings of the Conference on Software Maintenance* (CSM-87), (Austin, TX, September 1987).

[Freed] D.P. Freedman and G.M. Weinberg, *Handbook of Walkthroughs*, Inspections, and Technical Reviews, (Little, Brown, and Company, Boston, 1982).

[Glago] T.G. Glagowski, "Using a Relational Query Language as a Software Maintenance Tool," *Proceedings of the Conference on Software Maintenance* (CSM-85), (Washington, DC, November 1985).

[Grogo] P. Grogono, *Programming in Pascal*, (Addison-Wesley, Reading, MA, 1980).

[Hale] D.P. Hale and D.A. Haworth, "Software Maintenance: A Profile of Past Empirical Research," *Proceedings of the Conference on Software Maintenance* (CSM-88), (Phoenix, AZ, October, 1988).

[Hanse] K. Hansen, *Data Structured Program Design*, (Ken Orr and Associates, Topeka, KS, 1983).

[Haran] M.T. Harandi and J.Q. Ning, "PAT: A Knowledge-Based Program Analysis Tool," *Proceedings of the Conference on Software Maintenance* (CSM-88), (Phoenix, AZ, October, 1988).

[Horow] E. Horowitz, *Fundamentals of Programming Languages*, (Computer Science Press, MD, 1984).

[Huffm] J.E. Huffman and C. Burgess, "Partially Automated In-Line Documentation (PAID): Design and Implementation of a Software Maintenance Tool," *Proceedings of the Conference on Software Maintenance* (CSM-88), (Phoenix, AZ, October, 1988).

[Hunte] R. Hunter, *The Design and Construction of Compilers*, (John Wiley & Sons, NY, 1981).

[Jense] K. Jensen and N. Wirth, *Pascal User Manual and Report*, (Springer-Verlag, NY, 1974).

[Johns] S.C. Johnson, "Yacc: Yet Another Compiler Compiler," Computing Science Technical Report No. 32, (Murray Hill, NJ, 1975).

[Jones] C.B. Jones, *Software Development*, (Prentice-Hall International, London, 1980).

[Ker75] B.W. Kernigan, "Ratfor: A Preprocessor for a Rational FORTRAN," *Software Practice and Experience*, (1975).

[Ker78] B.W. Kernigan and D.M. Ritchie, *The C Programming Language*, (Prentice-Hall, NJ, 1978).

[Kuh85] D.R. Kuhn and C.G. Holls, "Simple Tools to Automate Documentation," *Proceedings of the Conference on Software Maintenance* (CSM-85), (Washington, DC, November 1985).

[Kuh87] D.R. Kuhn, "A Source Code Analyzer for Maintenance," Proceedings of the Conference on Software Maintenance (CSM-87), (Austin, TX, September 1987).

[Landi] L.D. Landis, P.M. Hyland, A.L. Gilbert and A.J. Fine, "Documentation in a Software Maintenance Environment," *Proceedings of the Conference on Software Maintenance* (CSM-88), (Phoenix, AZ, October, 1988).

[Lesk] M.E. Lesk, "The Portable C Library", *Computing Science Technical Report*, Report 31, (Murray Hill, NJ).

[Ma85a] J. Martin, *Fourth-Generation Languages*, Volumes 1-3, (Prentice-Hall, NJ, 1985).

[Ma85b] J. Martin and C. McClure, *Action Diagrams: Clearly Structured Program Design*, (Prentice-Hall, NJ, 1985).

[Ma85c] J. Martin and C. McClure, *Structured Techniques for Computing*, (Prentice-Hall, NJ, 1985).

[Ma87a] J. Martin, *Application Development Without Programmers*, (Prentice-Hall, NJ, 1987).

[Ma87b] J. Martin, *Recommended Diagramming Standards for Analysts and Programmers*, (Prentice-Hall, NJ, 1987).

[Meeke] J. Meekel and M. Viala, "LOGISCOPE: A Tool for Maintenance," *Proceedings of the Conference on Software Maintenance* (CSM-88), (Phoenix, AZ, October, 1988).

[Minsk] N.H. Minsky, "Controlling the Evolution of Large Scale Software Systems," *Proceedings of the Conference on Software Maintenance* (CSM-85), (Washington, DC, November 1985).

[Parik] G. Parikh and N. Zvegintzov, *Tutorial on Software Maintenance*, (IEEE Computer Society Press, Silver Springs, MD, 1983).

[Pau] L. Pau and J.M. Negret, "SOFTM: A Software Maintenance Expert System in PROLOG," *Proceedings of the Conference on Software Maintenance* (CSM-88), (Phoenix, AZ, October, 1988).

[Purdu] J.J. Purdum, T.C. Leslie and A.L. Stegemoller, *C Programmer's Library*, (Que Corporation, Carmel, IN, 1984).

[Rose] J.R. Rose, "Refined Types: Highly Differentiated Type Systems and Their Use in the Design of Intermediate Languages," *Proceedings of the SIGPLAN '88 Conference on Programming Language Design and Implementation*, (Association for Computing Machinery (ACM), NY, June 1988).

[Schrei] A.T. Schreiner and H.G. Friedman, Jr., *Introduction to Compiler Construction with UNIX*, (Prentice-Hall, NJ, 1985).

[Swann] G.H. Swann, *Top-Down Structured Design Techniques*, (Petrocelli Books, Princeton, NJ, 1978).

[Tausw] R.C. Tausworthe, *Standardized Development of Computer Science Software, Parts 1 and 2*, (Prentice-Hall, NJ, 1977).

[Terry] P.D. Terry, *FORTRAN From Pascal*, (Addison-Wesley, Reading, MA, 1987).

[Ullma] J.D. Ullman, *Principles of Database Systems*, (Computer Science Press, MD, 1982).

[Vick] C.R. Vick and C.V. Ramamoorthy, *Handbook of Software Engineering*, (Van Nostrand Reinhold Company, NY, 1984).

[Waite] W.M. Waite and G. Goos, *Compiler Construction*, (Springer-Verlag, NY, 1984).

[Warni] J. Warnier, *Logical Construction of Systems*, (Van Nostrand Reinhold Company, NY, 1981).

[Wedo] J.D. Wedo, "Structured Program Analysis Applied to Software Maintenance," *Proceedings of the Conference on Software Maintenance* (CSM-85), (Washington, DC, November 1985).

[Yourd] E. Yourdon and L.L. Constantine, *Structured Design*, (Prentice-Hall, NJ, 1979).

[Zelko] M.V. Zelkowitz, A.C. Shaw and J.D. Gannon, *Principles of Software Engineering and Design*, (Prentice-Hall, NJ, 1979).

[Zvega] N. Zvegintzov, ed., *Software Maintenance News*, (Staten Island, NY).

[Zve83] N. Zvegintzov, "Four Common Complaints - Tips Boost Maintenance Programmer Morale," *Tutorial on Software Maintenance*, G. Parikh and N. Zvegintzov, eds., (IEEE Computer Society Press, Silver Spring, MD, 1983).

[Zve88] N. Zvegintzov, "Attitude," *Software Maintenance News*, Volume 6, Number 8, (August 1988).